Scaling Analytics via Approximate and Distributed Computing

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Aniket Chakrabarti, BE, MS

Graduate Program in Computer Science and Engineering

The Ohio State University

2017

Dissertation Committee:

Srinivasan Parthasarathy, Co-Advisor

Christopher Stewart, Co-Advisor

Huan Sun

Wei Zhang

Abstract

The amount data generated from different sources everyday is increasing exponentially and for businesses to generate actionable insights, it is paramount that the analyses complete within a reasonable amount of time. Compute capacity, unfortunately, has not increased at a comparable rate. In recent past, researchers thus have focused on alternate approaches to scaling analytics to large datasets. Two of the most popular techniques used are (i) approximate computing techniques and (ii) distributed computing techniques for analytics execution speedup.

Approximate computing involves either a reduction in data size (usually through sampling and/or dimensionality reduction) or a reduction in algorithmic complexity. This results in significant gains in performance (in terms of execution time), however at a cost in reduction of accuracy for many analytics tasks. The biggest challenge in this space is understanding the tradeoff between performance and accuracy. Most of these approximate computing techniques do have some theoretical guarantees: for instance, the extremely popular principal component analysis (PCA) dimensionality reduction technique minimizes the data reconstruction error. However, a user typically interprets quality in terms of a few popular metrics such as recall and precision. A theoretical guarantee in terms of reconstruction error is not very useful from the context of a particular application of interest.

Distributed computing, on the other hand, tries to use many servers to process the data instead of a single server. Typical case is to partition the data across many servers process

the data partitions in parallel and combine the output from each server. This is the extremely popular MapReduce model of execution. However, at scale, it is likely that some machines will perform worse than others because they are slower, power constrained or dependent on undesirable, dirty energy sources. It is challenging to balance analytics workloads across heterogeneous machines because the algorithms are sensitive to statistical skew in data partitions. A skewed partition can slow down the whole workload or degrade the quality of results.

In the approximate computing space, we first begin by using the popular dimensionality reduction technique called locality sensitive hashing (LSH). In the context of one of the core analytics tasks, the all pairs similarity search (APSS), we design an algorithm that given a recall/precision target can automatically sketch the data using LSH (hence improving execution time) while guaranteeing the recall/precision target. The user does not need to explicitly set the sketch size and other LSH parameters. We then address the issue of generality. One key aspect of the APSS task is the similarity of interest (varies according to application domain). To generalize the APSS task across different similarity measures, we design a novel data embedding for supporting LSH on arbitrary kernel similarity measures (capable of representing any inner product similarity).

Prior works have mostly designed and developed approximation techniques and distributed computing techniques to scale up in isolation. We next show how insights from the approximate computing space are crucial in improving the execution of distributed analytics. We show that distributed analytics are extremely sensitive to the characteristics of data in the partitions. Key analytics tasks such as frequent pattern mining (FPM) degrades significantly in execution if the data partitions look statistically different (not representative of the true underlying distribution). We show the efficacy of LSH in creating a small sketch

to understand the global characteristics of the data and hence effectively partition the data of downstream analytics. Furthermore, we show that data approximation can also control the algorithmic complexity of graph analytics tasks. This is of particular interest as distributed graph analytics follow a different execution model (vertex-centric gather-update-scatter) that traditional tasks such as FPM (MapReduce model). We show that by approximating the input graph meaningfully we are able to significantly scale the Loopy Belief Propagation (LBP) inference to very large graphs. Finally, in the distributing computing space, we also developed a probabilistic method to manage latencies of distributed key value stores (usual storage medium for large data). This is crucial for interactive analytics.

Dedicated to my family

Acknowledgments

I would like to thank my advisors Srinivasan Parthasarathy and Christopher Stewart for their constant guidance, encouragement, and inspiration. I would also like to thank my committee members Huan Sun and Wei Zhang for their ideas and feedback. I'm grateful to my colleagues and collaborators at The Ohio State University for their valuable thoughts and frequent research discussions that made research a pleasant experience. I would like to thank my industry collaborators for bringing in new perspectives to my research. I'm grateful to my friends who made life at Columbus enjoyable. Finally, I'm indebted to my family for their constant encouragement, love, and support through out my doctoral studies.

My work is supported in part by NSF grants CNS-1230776, CCF-1217353, IIS-1141828, IIS-0917070, IIS-1111182, IIS-1111118, SHF-1217353, DMS-1418265, IIS-1550302, CCF-1629548, CNS-1350941, CSR-1320071 and CNS-1513120. The opinions and findings of this thesis are those of the author, and collaborators (where ever applicable) and do not necessarily reflect the views of the National Science Foundation or The Ohio State University.

Vita

August 2017	PhD, Computer Science and Engineering, The Ohio State University, USA.
December 2015	MS, Computer Science and Engineering, The Ohio State University, USA.
December 2007	BE, Information Technology, Jadavpur University, USA.

Publications

Research Publications

Aniket Chakrabarti, Srinivasan Parthasarathy, and Christopher Stewart. A Pareto Framework for Data Analytics on Heterogeneous Systems: Implications for Green Energy Usage and Performance. In *Proceedings of the 46th International Conference on Parallel Processing*, August 2017.

Yu Wang, Aniket Chakrabarti, David Sivakoff, and Srinivasan Parthasarathy. Fast Change Point Detection on Dynamic Social Networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, August 2017.

Yu Wang, Aniket Chakrabarti, David Sivakoff, and Srinivasan Parthasarathy. Hierarchical Change Point Detection on Dynamic Social Networks. In *Proceedings of the 9th International ACM Web Science Conference*, pages 171-179, June 2017.

Aniket Chakrabarti, Manish Marwah, and Martin Arlitt. Robust Anomaly Detection for Large-Scale Sensor Data. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, pages 31-40, November 2016.

Bortik Bandyopadhyay, David Fuhry, Aniket Chakrabarti, and Srinivasan Parthasarathy. Topological Graph Sketching for Incremental and Scalable Analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1231-1240, October 2016.

Aniket Chakrabarti, Bortik Bandyopadhyay, and Srinivasan Parthasarathy. Improving Locality Sensitive Hashing Based Similarity Search and Estimation for Kernels. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 641-656, September 2016.

Sávyo Toledo, Danilo Melo, Guilherme Andrade, Fernando Mourão, Aniket Chakrabarti, Renato Ferreira, Srinivasan Parthasarathy, and Leonardo Rocha. D-STHARk: Evaluating Dynamic Scheduling of Tasks in Hybrid Simulated Architectures. In *International Conference on Computational Science 2016*, pages 428-438, June 2016.

Aniket Chakrabarti, Srinivasan Parthasarathy, and Christopher Stewart. Green-and heterogeneity-aware partitioning for data analytics. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 366-371, April 2016.

Aniket Chakrabarti, Venu Satuluri, Atreya Srivathsan, and Srinivasan Parthasarathy. A bayesian perspective on locality sensitive hashing with extensions for kernel methods. In *ACM Transactions on Knowledge Discovery from Data (TKDD)*, Volume 10, Issue 2, Pages 19, October 2015.

Aniket Chakrabarti, and Srinivasan Parthasarathy. Sequential hypothesis tests for adaptive locality sensitive hashing. In *Proceedings of the 24th International Conference on World Wide Web*, pages 162-172, May 2015.

Christopher Stewart, Aniket Chakrabarti, and Rean Griffith. Zoolander: Efficiently Meeting Very Strict, Low-Latency SLOs. In *Proceedings of the 10th International Conference on Autonomic Computing*, pages 265-277, June 2013.

Aniket Chakrabarti, Christopher Stewart, Daiyi Yang, and Rean Griffith. Zoolander: efficient latency management in NoSQL stores. In *Middleware '12 Proceedings of the Posters and Demo Track*, Article No. 7, December 2012.

Fields of Study

Major Field: Computer Science and Engineering

Studies in:

Data Mining	Prof. Srinivasan Parthasarathy
Parallel and Distributed Computing	Prof. Christopher Stewart
Statistics	Prof. Douglas Critchlow

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Motivation

Advances in technology have enabled us to collect vast amounts of data across a myriad of domains for various purposes, ranging from astronomical observations to health screening tests, from computational fluid simulations to social network data, at an ever-increasing pace. To benefit from these large and complex data stores, it is clear that methods are needed to efficiently analyze the data to facilitate effective data understanding and knowledge discovery since a lengthy time delay between responses of two consecutive user requests can disturb the flow of human perception and formation of insight. Unfortunately, the processor clock speeds have saturated, leading to an impedance mismatch. For a long time processor clock speeds were able to keep up with the rate of increase in data because of Dennard scaling [52]. The effect of Dennard scaling was that the number of transistors on a chip doubled every 18-24 months (Moore's law) resulting in proportional clock speed increase. The current impedance mismatch arises because Dennard scaling for single core and multicore machines has ended [55] due to power constraints.

Two of the most popular techniques for scaling analytics to large datasets are (i) approximate computing techniques and (ii) distributed computing techniques.

Approximate computing techniques usually involve generating a small sketch of the large data that captures the key features and then running analytics on the sketch. Sampling [158] and dimensionality reduction [58] are two of the important techniques of data size reduction. Data sketching implies significant data size reduction resulting in improved execution times of the analytics tasks. However, this improvement of execution time comes at cost of compromising the task quality. Another area of approximate computing is to simplify the computational complexity of the algorithm, hence improving performance, again at a cost of loss in quality. Hence it is paramount to understand the performance-accuracy tradeoff with respect to using approximate computing techniques. More importantly, this tradeoff has to be interpretable by the application user. For instance, PCA, a de-facto dimensionality reduction technique comes with the theoretical guarantee of minimizing data reconstruction error. However, typically application user interprets the quality of a task in terms recall and precision and it is unclear how those metrics of interest behave when approximate computing techniques are applied.

Distributed computing on the other hand partitions the large input data into smaller partitions and each partition is independently processed by a separate server and the output of all the servers is combined to generate meaningful insight. This is the MapReduce [49] computing model. Under this model, to scale a task to large datasets, simply the addition of servers is sufficient. However, in practice, scaling analytics tasks using distributed computing techniques face additional challenges as the datacenters and supercomputing centers of today are increasingly becoming a hive of heterogeneous technologies. Hardware from multiple chip manufacturers, storage solutions, and software from multiple vendors, dozens of unique applications and data management solutions and tools are fairly common place. There are many reasons for such heterogeneity such as equipment upgrades and power-performance

tradeoffs within modern processor families. Leveraging such a heterogeneous computing environment effectively is challenging – given the scales of data involved and furthermore given the irregularity and dependency of the runtime of such algorithms on payload content (not just size). The demand for computational resources to solve such big data problems is increasing. At the same time, the increasing demand for computational power comes at the cost of higher aggregate energy consumption. The computers' insatiable appetite for energy has created an IT industry that has approximately the same carbon footprint as the airline industry [27], accounting for a significant fraction of the global emissions. To make matters worse, energy consumption is quickly becoming a limiter for big data science. Building an exascale machine with today's technology is impractical due to the inordinate power draw it would require, hampering large-scale scientific efforts. Governments organizations and agencies have begun to incentivize consumers and industry with a variety of tax credits, rebates and other incentives to support energy efficiency and encourage the use of renewable energy sources when available. Further complicating this issue, as observed by several researchers is the fact that there is often heterogeneity in the availability of green energy within a data center [51, 99, 171].

1.2 Thesis Statement

The statement of this thesis is that scaling analytics tasks to large datasets requires an effective coupling of approximate computing algorithms along with appropriate distributed systems support. Specifically, in the approximate computing domain, we analyze the problem of providing application specific quality guarantees and generalizing the tasks to multiple application domains using principled approximate data sketching techniques. In the distributed computing space, we examine the impact of payload (data) characteristics on the

performance and quality of the underlying analytics tasks and further design a framework that can partition and place data in a payload-aware manner guided by the approximate sketch of the payload.

1.3 Contributions

In this thesis, we propose to scale up analytics tasks to large datasets using dimensionality reduction based data sketching techniques with rigorous quality guarantees. Furthermore, we show that sketching based data characterization can play a key role in data partitioning and placement in distributed analytics. Specifically, the contributions of this thesis are as follow:

- We solve the APSS problem that is a core analytics kernel using approximate computing techniques. We use LSH, a popular dimensionality reduction technique to scale APSS to very large datasets on a single server. Our method requires as input a recall target (intuitive quality metric for APSS application) and automatically sketches the input data using LSH to achieve that recall while reducing execution time as a result of the small sized sketch.

- We deal with the problem of generalizing APSS application to a wide variety of application domains. We achieve this generalization by designing unbiased techniques of extending LSH to support any arbitrary kernel similarity measures. Such inner product similarity measures are known to work extremely well in complex application domains such as images, protein sequences, and text.

- We design a data approximation technique that can significantly improve the convergence of LBP, one of the standard inference techniques for Markov models. We use distributed anomaly detection as an application.

- We propose a holistic framework for distributed analytics. This framework is built as a middleware on top of NoSQL stores and is capable of partitioning the data for downstream analytics in a performance, energy, and payload aware way.

- We investigate the underlying distributed storage substrate that is amenable to distributed interactive analytics workloads. We propose probabilistic model based approach for latency management on distributed NoSQL stores using replication and sharding.

1.4 Overview of Solutions

In this section, we provide an overview of the methods we proposed for each of our contributions. As we described earlier, our proposed solutions for scaling analytics can broadly be divided into two categories, (i) approximate computing methods and (ii) distributed computing techniques.

1.4.1 Approximate Computing through Locality Sensitive Hashing

Locality Sensitive Hashing is a technique to sketch (summarize) data such that similar data have similar sketches with high probability. This makes LSH an excellent candidate for summarizing huge datasets because data sketches reduce the dataset size and analytics tasks that rely on pairwise similarity can be run on the sketch instead of the original data. Of course, since the technique is probabilistic, there will be a loss of accuracy as the data sketch size is reduced. One significant challenge in this space is to understand what is the

required sketch size to ensure certain quality (in terms of common analytics measures such as precision, recall etc.). We address this challenge for one of the core analytics tasks – All Pairs Similarity Search Problem (APSS).

LSH for APSS [30, 33]: Similarity search in a collection of objects has a wide variety of applications such as clustering, semi-supervised learning, information retrieval, query refinement on web search, near duplicate detection, collaborative filtering, and link prediction. Formally, the problem statement is: Given a collection of objects D and an associated similarity measure $sim(.,.)$ and a similarity threshold t, the problem is to find all pairs of objects x, y, such that $sim(x, y) \geq t$, where $x, y \in D$. The major challenge here is dealing with the large volume of data. The volume combined with high dimensionality of datasets can lead to inefficient solutions to this problem. Traditional solution to the problem includes two steps: (i) candidate pair generation and (ii) similarity computation. Recent research has focused on reducing the candidate search space. The exact solution in this space, the AllPairs candidate generation algorithm, builds a smart index structure that prunes out a lot of candidate pairs from the vector representation of the points and computes exact similarity. The obvious advantage of using LSH instead is that it allows (i) to build an index that reduces the candidate search space (still has a significant number of false positives) and (ii) it leads to an unbiased estimator of similarity, inference on which allows further reduction of the search space by pruning out a large number of the aforementioned false positives. Hence LSH based techniques can significantly improve the performance of both steps 1 and 2 for the APSS problem, however, the quality of the application (recall) will degrade with the reduction of sketch size and it is hard for a user to guess the correct sketch size. Our contribution is to develop an algorithm that given a quality parameter, finds the required sketch size to rigorously guarantee the quality.

The idea is to incrementally generate and compare batches of hash functions for a pair of points x, y and stop when according to some model it can be said with certainty whether $sim(x,y) \geq t$. We model the problem as a sequential hypothesis test problem and provide rigorous quality (recall) guarantees through Type I error. We start with the traditional Sequential Probability Ratio Test (SPRT) and show that it is extremely inefficient in practice, making it unsuitable for large scale analytics. We then propose an alternate sequential hypothesis test procedure based on a one-sided fixed-width confidence limit construction technique. Finally, we show that no single hypothesis testing strategy works well for all similarity values. Therefore, we propose a fine-grained hybrid hypothesis testing strategy, that based on a crude estimate of the similarity from the first batch of hash comparisons for that specific candidate pair selects the most suitable test for that pair. In other words, instead of using a single hypothesis test, we dynamically choose the best suited hypothesis test for each candidate pair. We extend the above technique to develop a variant, that after candidate pruning, estimates the approximate similarity by using a fixed-width two-sided confidence interval generation technique.

Generalizing LSH to Kernels [28]: One of the advantages of LSH is that such hash functions exist for a number of different similarity measures such as Jaccard and cosine similarities. Recently kernel based similarity measure have found increased use in complex domains such as images, text, protein sequences etc. in part because the data becomes easily separable in the kernel induced feature space. However, the kernel functions are usually extremely computationally intensive and to make things worse, explicit vector representation of the data points in the kernel induced feature space is usually infinite dimensional. Hence APSS methods requiring explicit representation do not work making LSH all the more important for kernel spaces. Efficient and accurate kernel approximation

techniques either involve the kernel principal component analysis (KPCA) approach or the Nyström approximation method. We show that extant LSH methods in this space suffer from a bias problem, that moreover is difficult to estimate apriori. Consequently, the LSH estimate of a kernel is different from that of the KPCA/Nyström approximation. We provide a theoretical rationale for this bias, which is also confirmed empirically. We propose an LSH algorithm by projecting the data points to a new embedding that can reduce this bias and consequently our approach can match the KPCA or the Nyström methods' estimation accuracy while retaining the traditional benefits of LSH.

1.4.2 Distributed Computing for Scaling Analytics

Distributed Anomaly Detection in Sensor Networks [29]: Large scale sensor networks are ubiquitous nowadays. An important objective of deploying sensors is to detect anomalies in the monitored system or infrastructure, which allows remedial measures to be taken to prevent failures, inefficiencies, and security breaches. Most existing sensor anomaly detection methods are local, i.e., they do not capture the global dependency structure of the sensors, nor do they perform well in the presence of missing or erroneous data. We proposed an anomaly detection technique for large scale sensor data that leverages relationships between sensors to improve robustness even when data is missing or erroneous. We develop a probabilistic graphical model based global outlier detection technique that represents a sensor network as a pairwise Markov Random Field (MRF). We use the Loopy Belief Propagation (LBP) inference technique on MRFs. Since LBP is computationally intensive we used the GraphLab framework to scale it to large networks. Additionally, we show that approximating the network graph structure meaningfully can significantly improve the convergence rate of LBP. We did a detailed study on both distributed (across machines)

and parallel (multicores in a single machine) using the GraphLab. We also contrasted the performance of asynchronous execution versus synchronous execution. We scaled up to network size of 120 million edges.

Heterogeneity-Aware Data Partitioning [31]: Distributed algorithms for analytics partition their input data across many machines for parallel execution. At scale, it is likely that some machines will perform worse than others because they are slower, power constrained or dependent on undesirable, dirty energy sources. It is even more challenging to balance analytics workloads as the algorithms are sensitive to statistical skew across data partitions and not just partition size. Here, we propose a lightweight framework that controls the statistical distribution of each partition and sizes partitions according to the heterogeneity of the environment. We model heterogeneity as a multi-objective optimization, with the objectives being functions for execution time and dirty energy consumption. We use stratification to control data skew and model heterogeneity-aware partitioning. We then discover Pareto optimal partitioning strategies. We built our partitioning framework atop Redis and measured its performance on data mining workloads with realistic data sets. We extend our partitioning strategy to integrate it with Spark and Hadoop such that the data partitions within those frameworks are skew aware.

Latency management for NoSQL stores [144]: Internet services access networked storage many times while processing a request. Just a few slow storage accesses per request can raise response times a lot, making the whole service less usable and hurting profits. We proposed Zoolander, a key value store that meets strict, low latency service level objectives (SLOs). Zoolander scales out using replication for predictability, an old but seldom-used approach that uses redundant accesses to mask outlier response times. Zoolander also scales

out using traditional replication and partitioning. It uses an analytic model to efficiently combine these competing approaches based on systems data and workload conditions. For example, when workloads under utilize system resources, Zoolander's model often suggests replication for predictability, strengthening service levels by reducing outlier response times. When workloads use system resources heavily, causing large queuing delays, Zoolander's model suggests scaling out via traditional approaches. We used a diurnal trace from a popular online service to test Zoolander at scale (up to 40M accesses per hour) on Amazon EC2. We tested Zoolander on a number of different NoSQL stores: Cassandra, Zookeeper, and Redis.

1.5 Outline

The thesis is organized as follows. In Chapter 2 we describe our LSH based APSS algorithm. Next in Chapter 3 we design a novel data embedding that allows an unbiased LSH estimator for arbitrary kernel similarity measures. In Chapter 4 we describe a data approximation technique that can significantly speed up the convergence of LBP for anomaly detection. Then in Chapter 5 we develop a distributed framework for analytics that takes into account the data characteristics (generated by LSH) and also takes into account the processing capacity heterogeneity of modern data center environments. In Chapter 6 we investigate the issue of distributed storage required for realtime analytics on large datasets. Finally, we conclude in chapter 7 by summarizing the contributions of this thesis followed by the open challenges and a future roadmap to address those.

Chapter 2: Scaling All Pairs Similarity Search using Adaptive Locality Sensitive Hashing

In this chapter, we seek to address one of the core kernels of analytics, the *All Pairs Similarity Search* problem through *Approximate Computing* techniques. Specifically, we propose to use the *Locality Sensitive Hashing* dimensionality reduction technique to create a small sketch of the large input data and develop an APSS algorithm for the sketch. The key challenge we address is to rigorously guarantee the quality metric (recall) required by the APSS algorithm by connecting it to the probabilistic guarantees provided by LSH, while automatically generating the LSH sketch.

2.1 Introduction

Similarity search in a collection of objects has a wide variety of applications such as clustering [127], semi-supervised learning [174], information retrieval [59], query refinement on websearch [18], near duplicate detection [161], collaborative filtering, and link prediction [100]. Formally, the problem statement is: *Given a collection of objects D and an associated similarity measure* $sim(.,.)$ *and a similarity threshold t, the problem is to find all pairs of objects* x, y*, such that* $sim(x,y) \geq t$*, where* $x, y \in D$.

The major challenge here is dealing with the large volume of data. The volume combined with high dimensionality of datasets can lead to inefficient solutions to this problem. Recent

research has focused on reducing the candidate search space. The AllPairs [18] candidate generation algorithm builds a smart index structure from the vector representation of the points to compute exact similarity. Another approach is to do an approximate similarity calculation that involves sacrificing small accuracy for substantial speedup. The most popular technique is locality sensitive hashing (LSH) [65, 79] that involves projecting the high dimensional data into a lower dimensional space and similarity of a pair of points is approximated by the number of matching attributes in the low dimensional space.

A recent idea, BayesLSHLite [131], adopts a principled Bayesian approach on top of LSH and AllPairs to reason about and estimate the probability that a particular pair of objects will meet the user-specified threshold. Unpromising pairs are quickly pruned away based on these estimates realizing significant performance gains. The way BayesLSHLite works is, for a pair of data objects x, y, BayesLSHLite incrementally compares their hashes (in batches of size b) and infers after each batch, how likely is it that pair will have a similarity above the threshold. If that probability becomes too low, then the candidate pair is pruned away. The above algorithm has a variant called BayesLSH, where after the candidate pruning is done, the similarity is estimated approximately from the hash signatures instead of exact computation. This is useful in cases where exact similarity computation is infeasible. Exact similarity computation might be infeasible in cases where the original data was too large to store and the small hash sketch of the data was stored instead. Another scenario could be where the similarity measure-of-interest is a kernel function and exact representation of the data points in the kernel induced feature space is not possible as that space might be infinite dimensional (e.g. Gaussian RBF kernel). Additionally, the specialized kernel functions are extremely expensive to compute on the fly. In such cases, both candidate generation and similarity estimation have to be done approximately using the LSH hash sketches.

In this chapter, we adapt a Frequentist view of this idea and propose a fully principled sequential model to do the incremental pruning task with rigorous quality guarantees. Specifically, we model the problem of $sim(x,y) \geq t$ as a sequential hypothesis test problem and provide quality guarantees through Type I and Type II errors. We start with the traditional SPRT [153] and show that it is extremely inefficient in practice, making it unsuitable for large scale analytics. We then propose an alternate sequential hypothesis test procedure based on a one-sided fixed-width confidence limit construction technique. Finally, we show that no single hypothesis testing strategy works well for all similarity values. Therefore, we propose a fine-grained hybrid hypothesis testing strategy, that based on a crude estimate of the similarity from the first batch of hash comparisons for that specific candidate pair selects the most suitable test for that pair. *In other words, instead of using a single hypothesis test, we dynamically choose the best suited hypothesis test for each candidate pair.* We extend the above technique to develop a variant, that after candidate pruning estimates the approximate similarity by using the fixed-width two-sided confidence interval generation technique [61]. Our ideas are simple to implement and we show that our hybrid method always guarantees the minimum quality requirement (as specified by parameters input to the algorithm), while being up to 2.1x faster than an SPRT-based approach and 8.8x faster than AllPairs while qualitatively improving on the state-of-the-art BayesLSH/Lite estimates.

2.2 Background

2.2.1 Locality Sensitive Hashing

Locality sensitive hashing [65, 79] is a popular and fast method for candidate generation and approximate similarity computation within a large high dimensional dataset. Research has demonstrated how to leverage the key principles for a host of distance and similarity

measures [25, 26, 45, 73, 79, 127]. Briefly, each data point is represented by a set of hash keys using a specific hash family for a given distance or similarity measure (*sim*). Such a family of hash function is said to have the locality sensitive hashing property if:

$$P_{h \in F}(h(x) == h(y)) = sim(x, y) \tag{2.1}$$

where x, y are any two points in the dataset, and h is a randomly selected hash function from within a family F of hash functions. Consequently, the approximate similarity between the pair can be estimated as:

$$\hat{sim}(x, y) = \frac{1}{n} \sum_{i=1}^{n} I[h_i(x) == h_i(y))]$$

where n is the total number of hash functions and I is the indicator function.

2.2.2 Candidate generation

We use AllPairs [18] candidate generation algorithm when the original data set is available. The AllPairs candidate generation algorithm is exact, hence all true positives (candidates with similarity above the threshold) will be present in the set of candidates generated. The AllPairs algorithm builds an index from the vector representation of data and instead of building a full inverted index, AllPairs only indexes the information that may lead to a pair having a similarity greater than the specified threshold. The AllPairs algorithm can be used when the original dataset is small enough to be stored entirely and the similarity of interest is computed on the original feature space (unlike kernel similarity measures).

In cases where the entire dataset cannot be stored, rather a small sketch of it is available, AllPairs cannot be used. Additionally, if the similarity is a kernel function and the feature space of that kernel cannot be explicitly represented, AllPairs will not work as it relies on the explicit representation. However, in both scenarios, we can use LSH to generate

a low dimensional sketch of the dataset and use the probabilistic candidate generation algorithms [25, 26, 45, 73, 79, 127] to build the index. The advantage of such an index structure is that for a query point, similarity search can be done in sublinear time. LSH based index structures perform well compared to traditional indices when the dimensionality of the data set is very high. The algorithm is as follows:

1. *Using an LSH method for a given similarity measure, form l signatures for each data point, each signature having k hash keys.*

2. *Each pair of points that share at least one signature will stay in the same hash bucket.*

3. *During all pairs similarity search, for each point, only those points which are in the same bucket needs to be searched.*

4. *From [161], for a given k and similarity threshold t, the number of signatures l required for a recall* $1-\phi$*,*

$$l = \lceil \frac{log(\phi)}{log(1-t^k)} \rceil$$

2.2.3 Candidate Pruning using BayesLSH/Lite

Traditionally maximum likelihood estimators are used to approximate the similarity of a candidate pair. For the candidate pair, if there are a total of n hashes and m of them match, then the similarity estimate is $\frac{m}{n}$. The variance of this estimator is $\frac{s(1-s)}{n}$ where s is the similarity of the candidate pair. Two issues to observe here are - 1) as n increases, the variance decreases and hence the accuracy of the estimator increases, 2) more importantly, the variance of the estimator depends on the similarity of the pair itself. This means for a fixed number of hashes, the accuracy achieved if the similarity of the candidate pair was 0.9 is higher than if the similarity was 0.5. In other words, the number of hashes

required for different candidate pairs for achieving the same level of accuracy is different. Therefore, the problem with fixing the number of hashes is - some of the candidate pairs can be pruned by comparing the first few hashes only. For example, if the similarity threshold is 0.9 and 8 out of the first 10 hashes did not match, there is very low probability that the similarity of the pair is greater than 0.9. The BayesLSH/Lite [131] is among the earliest approaches to solving this problem of fixing the number of hashes. Instead, it incrementally compares hashes until the candidate pair can be pruned with a certain probability, or the maximum allowed hash comparisons are reached. It will then compute exact or approximate similarity. To do the incremental pruning, BayesLSHLite solves the inference problem $P(sim(x,y) \geq t)$, where t is the similarity threshold. Additionally, the BayesLSH variant estimates the approximate similarity by creating an interval for the true similarity by the solving the inference $P(|sim(x,y) - sim(\hat{x},y)| \leq \delta)$. Both inferences are solved using a simple Bayesian model. These inferences help overcome the problem pointed above - number of hashes required to prune a candidate or build an interval around it adaptively set for each candidate pair.

Let us denote the similarity $sim(x,y)$ as the random variable S. Since we are counting the number of matches m out of n hash comparison, and the hash comparisons are i.i.d. with match probability S as per equation 2.1, the likelihood function becomes a binomial distribution with parameters n and S. If $M(m,n)$ is the random variable denoting m matches out of n hash bit comparisons, then the likelihood function will be:

$$P(M(m,n)|S) = \binom{n}{m} S^m (1-S)^{n-m} \tag{2.2}$$

In the Bayesian setting, the parameter S can be treated as a random variable. Let the estimate of S in this setting be $\hat{S}$. Using the aforementioned likelihood function, the two inference problems become:

Early pruning inference: given m matches out of n bits, what is the probability that the similarity is above threshold t:

$$P[S \geq t \,|\, M(m,n)] \;=\; \int_t^1 P(S \,|\, M(m,n))dS \tag{2.3}$$

Concentration inference: given the similarity estimate $\hat{S}$, what is the probability that it falls within δ of the true similarity:

$$\begin{aligned} P[|S-\hat{S}| < \delta \,|\, M(m,n)] &= P[\hat{S}-\delta < S < \hat{S}+\delta \,|\, M(m,n)] \\ &= \textstyle\int_{\hat{S}-\delta}^{\hat{S}+\delta} P(S \,|\, M(m,n))dS \end{aligned} \tag{2.4}$$

The BayesLSHLite algorithm works as follows:

For each pair x,y:

1. *Compare the next b hashes and compute the early pruning probability* $P[S \geq t \,|\, M(m,n)]$.
2. *If* $P[S \geq t \,|\, M(m,n)] < \alpha$, *then prune the pair and stop.*
3. *If maximum allowable hash comparisons have been reached, compute exact similarity and stop.*
4. *Go to step 1.*

The BayesLSH variant works as follows:

For each pair x,y:

1. *Compare the next b hashes and compute the early pruning probability* $P[S \geq t \,|\, M(m,n)]$.
2. *If* $P[S \geq t \,|\, M(m,n)] < \alpha$, *then prune the pair and stop.*
3. *If* $P[|S-\hat{S}| < \delta \,|\, M(m,n)] > 1-\gamma$, *then output pair* x,y *if* $\hat{S} \geq t$ *and stop.*

4. *If maximum allowable hash comparisons have been reached, then output pair x, y if $\hat{S} \geq t$ and stop.*

5. *Go to step 1.*

2.3 Case for Frequentist Formulation

The BayesLSH/Lite candidate pruning and approximate similarity estimation algorithms as described in the previous section, provide the basis for the current work. Specifically, in this work, we examine the same problems they attempt to solve but in a Frequentist setting. BayesLSH/Lite makes a simplifying assumption potentially leading to somewhat weak bounds on error. *Specifically, BayesLSH/Lite tries to model an inherently sequential decision process in a non-sequential way.* The inferences (equations 2.3 and 2.4) in the BayesLSH/Lite algorithms are done every b hash comparisons (this can be viewed as a bin of comparisons). Therefore, for a candidate pair, if the pruning inference is done once, then the error probability rate will α, but when it is done for the second time, probability of the pair getting pruned will be determined by getting pruned the first time (first bin) and the probability of getting pruned the second time (a cumulative of the first and second bin matches). Essentially, we argue that this error may propagate resulting in an accumulated error over multiple pruning inferences. The same scenario is true for the concentration inference as well (equation 2.4). Over multiple concentration inferences done incrementally, the coverage probability could fall below $1 - \gamma$. We note that in practice this may not be a significant issue but the question remains can this problem be fixed (in the rare cases it may materialize) without significantly impacting the gains obtained by BayesLSH/Lite. We note that fixing this problem in a Bayesian setting remains open but in this work, we show how this problem can be fixed to guarantee tighter error bounds in a Frequentist setting.

Another issue, again a minor one, is when a pair is unlikely to be above a certain similarity threshold, pruning it early saves hash comparisons, similarly when a pair is very likely to be above the threshold, hash comparison for it should stop immediately and it should be processed for exact/approximate similarity computation. This can also save a number of hash comparisons.

To overcome these problems, we propose to model the problem in a Frequentist setting as follows. In the frequentist setting let the similarity $sim(x,y)$ be denoted by the parameter s (instead of S as in Bayesian setting).

- We model the early pruning inference $s > t$ as a *sequential hypothesis test* problem that should be able to guarantee Type I and Type II errors under the sequential hash comparison setting and if possible, it should be able to early prune a pair or send a pair for exact/approximate similarity computation.

- We model the concentration inference $|s - \hat{s}| \leq \delta$ as a *sequential two-sided fixed-width confidence interval* creation problem that should be able to guarantee a certain coverage probability.

2.4 Methodology

In this section, we describe a principled way of doing the early pruning inference (equation 2.3) and the concentration inference (equation 2.4) under the sequential setting where the number of hash functions (n) is not fixed, rather it is also a random variable.

2.4.1 Early Pruning Inference

We use sequential tests of composite hypothesis for pruning the number of generated candidates, so that the cardinality of the remaining candidate set is very small. Therefore,

exact similarity computation on the remaining set of candidate pairs becomes feasible in terms of execution time, provided the original data set is available and the similarity function can be computed on the feature space of the original data. Our pruning algorithm involves sequentially comparing the hashes for a pair of data objects and stop when we are able to infer with some certainty whether the similarity for the pair is above or below the user defined threshold. If, according to the inference, the similarity of the pair is below the threshold, then we prune away the pair, otherwise, we compute the exact or approximate similarity of the pair depending on which variant we are using. More formally, if the similarity of the pair is s and the user defined threshold is t, we need to solve the hypothesis test, where the null hypothesis is $H0 : s \geq t$ and the alternate hypothesis is $H1 : s < t$. Two interesting aspects of our problem formulation are:

1. For performance reasons as described in section 2.3, we do not want to fix the number of hashes to compare for the hypothesis test, but rather incrementally compare the hashes and stop when a certain accuracy in terms of Type I error has been achieved.

2. We focus on Type I error, i.e. we do not want to prune away candidate pairs which are true positives ($s \geq t$). We can allow false positives ($s < t$) in our final set, as either exact similarity computation or approximate similarity estimation will be done on the final set of candidate pairs and any false positives can thus be pruned away. In other words, we do not need to provide guarantees on Type II error of the hypothesis test. Of course keeping a low Type II error implies less false positives to process, resulting in better performance.

We discuss three strategies for formulating the hypothesis test. First, we cast our problem in a traditional Sequential Probability Ratio Test (SPRT) [153] setting and then discuss the

shortcomings of such an approach. Second, we then develop a sequential hypothesis test based on a sequential fixed width one-sided confidence interval (CI) and show how it can overcome some of the limitations of traditional SPRT. We empirically find that even this test does not always perform better than the more traditional SPRT. Third, building on the above, we propose a hybrid approach (HYB), where we dynamically select the hypothesis test (SPRT or CI) based on the similarity of each candidate pair which we crudely estimate from the first few comparisons. In other words, *instead of using a single fixed hypothesis test, we select one which is best suited for the candidate pair being estimated.*

For a candidate pair x, y with similarity s, the probability of a hash matching is s for a locality sensitive hash function as described in equation 2.1. Therefore, given n hashes for the pair, the probability that m of them will match follows a binomial distribution with parameters n and s. This is because the individual hash matching probabilities are identically and independently distributed Bernoulli with parameter s. So our problem formulation reduces to doing sequential hypothesis test on a binomial parameter s.

2.4.1.1 Sequential Probability Ratio Test

We use the traditional sequential probability ratio test by Wald [153] as our first principled sequential model for matching LSH signatures, to decide between $s \geq t$ or $s < t$. For the purpose of this model we swap the null and alternate hypotheses of our formulation. We do this because the resulting formulation of the hypothesis test $H0 : s < t$ vs. $H1 : s \geq t$, where s is a binomial parameter, has a well known textbook solution (due to Wald). The important thing to recollect is that we care more about Type I error in our original formulation. Therefore under the swapped SPRT setting, we care about the Type II error. That is easily done as SPRT allows the user to set both Type I and Type II errors, and we set Type II error to be α. To solve a composite hypothesis test using SPRT for a binomial parameter

s, the first step is to choose two points $t+\tau$ and $t-\tau$. Now the SPRT becomes a simple hypothesis test problem of $H0 : s = s_0 = t-\tau$ vs. $H1 : s = s_1 = t+\tau$. The algorithm works as follows:

1. Incrementally compare batches of size b hashes until

$$\frac{log(\frac{\alpha}{1-\beta})}{log(\frac{s_1}{s_0}) - log(\frac{1-s_1}{1-s_0})} + n\frac{log(\frac{1-s_0}{1-s_1})}{log(\frac{s_1}{s_0}) - log(\frac{1-s_1}{1-s_0})} < \hat{s}$$
$$< \frac{log(\frac{1-\alpha}{\beta})}{log(\frac{s_1}{s_0}) - log(\frac{1-s_1}{1-s_0})} + n\frac{log(\frac{1-s_0}{1-s_1})}{log(\frac{s_1}{s_0}) - log(\frac{1-s_1}{1-s_0})}$$

   Here n is the cumulative number of hash comparisons till now, and $\hat{s} = m/n$, where m is the cumulative number of hash matches up to that point.

2. Reject null hypothesis (conclude $s \geq t$) if,

$$\hat{s} \geq \frac{log(\frac{1-\alpha}{\beta})}{log(\frac{s_1}{s_0}) - log(\frac{1-s_1}{1-s_0})} + n\frac{log(\frac{1-s_0}{1-s_1})}{log(\frac{s_1}{s_0}) - log(\frac{1-s_1}{1-s_0})}$$

3. Fail to reject null hypothesis (conclude $s < t$) if,

$$\hat{s} \leq \frac{log(\frac{\alpha}{1-\beta})}{log(\frac{s_1}{s_0}) - log(\frac{1-s_1}{1-s_0})} + n\frac{log(\frac{1-s_0}{1-s_1})}{log(\frac{s_1}{s_0}) - log(\frac{1-s_1}{1-s_0})}$$

SPRT is a cumulative likelihood ratio test and is an optimal test with guaranteed Type I and Type II errors when the hypotheses are simple. In the case of a composite hypothesis (across bins of hashes), no optimality guarantees can be given, and consequently, to make a decision, SPRT typically takes a large number of hash comparisons. This results in extremely slow performance as we will empirically validate. We next describe the confidence interval based test.

2.4.1.2 One-Sided-CI Sequential Hypothesis Test

2.4.1.2.1 Constructing the confidence interval (CI) The true similarity s of pair of data objects x, y can be estimated as $\hat{s} = \frac{m}{n}$, where m is the number of hashes that matched

out of n hash comparisons. It can be shown that $\hat{s}$ is the maximum likelihood estimate of s [128]. Following standard convention, we denote this estimator as $\hat{S}$ (random variable, distinguished from its realization, $\hat{s}$). Here we describe the procedure for constructing a fixed-width (say w) upper confidence interval for s with $1-\alpha$ coverage probability. More formally, we want to continue comparing hashes and estimating similarity until,

$$P(s < \hat{S} + w) = 1 - \alpha \tag{2.5}$$

Here $\hat{s} + w$ is the upper confidence limit for s with $1-\alpha$ coverage probability. We use an approach similar to Frey [61] to solve this problem.

Stopping rule for incremental hash comparisons: We use the Wald confidence interval for binomial as our stopping rule. Formally, for some value λ, and a fixed confidence width w, we incrementally compare batches of b hashes and stop when $z_\lambda \sqrt{\frac{\hat{s_a}(1-\hat{s_a})}{n}} \leq w$. Then the upper confidence limit can be reported as $min(\hat{s} + w, 1.0)$. Here $\hat{s_a} = \frac{m+a}{n+2a}$, where a is a very small number. $\hat{s_a}$ is used instead of $\hat{s}$ because, if the batch size is extremely small, and number of matches is 0 (or it is the maximum, i.e., all match), then the confidence width becomes 0 after the first batch if $\hat{s}$ is used.

Finding λ: In a non-sequential setting, the Wald upper confidence limit as described above will have a coverage probability of $1-\lambda$. But in a sequential setting, where the confidence interval is tested after every batch of hash comparisons, the coverage probability could fall below $1-\lambda$. Hence to ensure coverage probability of at least $1-\alpha$, λ should be set less than α. Given the set of stopping points and a λ, we can compute the coverage probability CP(λ) of our one-sided confidence interval using the path-counting technique [66]. Suppose the stopping points are $(m_1, n_1), (m_2, n_2), ..., (m_k, n_k)$ and $H(m,n)$ is the number of ways to do n hash comparisons with m matches, without hitting any of the stopping points. Therefore,

the probability of stopping at stopping point (m_i, n_i) is $H(m_i, n_i)s^{m_i}(1-s)^{n_i-m_i}$. Since the incremental hash comparison process is guaranteed to stop, probability of stopping at all the stopping points should sum to 1. This implies

$$\sum_{i=1}^{k} H(m_i, n_i)s^{m_i}(1-s)^{n_i-m_i} = 1$$

Consequently, the coverage probability for similarity s will be

$$T(s,\lambda) = \sum_{i=1}^{k} H(m_i, n_i)s^{m_i}(1-s)^{n_i-m_i} I(s \leq \frac{m_i}{n_i} + w) \tag{2.6}$$

Here I is the indicator function of whether the stopping point is within the correct interval width w. Now the overall coverage probability can be computed as,

$$CP(\lambda) = min_{s\in[0,1]} T(s,\lambda) \tag{2.7}$$

For our one-sided confidence interval to have at least $1-\alpha$ coverage probability, we need to find λ such that $CP(\lambda) \geq 1-\alpha$. The function $H(m,n)$ can be solved using the path-counting recurrence relation:

$$\begin{aligned} H(m,n+1) =& H(m,n)ST(m,n) \\ &+ H(m-1,n)ST(m-1,n) \end{aligned}$$

Here $ST(m,n)$ is the indicator function of whether (m,n) is a non-stopping point. The base of the recursion is $H(0,1) = H(1,1) = 1$. With a fixed λ, $ST(m,n)$ can be computed using the Wald stopping rule as described before, and $H(m,n)$ can be hence computed by the aforementioned recurrence relation. Then we need to solve equation 2.7 to find out the confidence coefficient of our one-sided interval. $T(s,\lambda)$ is a piecewise polynomial in s with jumps at the points in the set $C = \{0, \frac{m_i}{n_i} + w, \forall i = 1 \; to \; k \; and \; \frac{m_i}{n_i} + w \leq 1\}$. $CP(\lambda)$ is then approximated numerically by setting $s = c \pm 10^{-10}$, where $c \in C$ and taking the minimum resulting $T(s,\lambda)$. Now that we know how to compute $CP(\lambda)$, we use bisection root-finding algorithm to find a λ for which $CP(\lambda)$ is closest to our desired coverage probability $1-\alpha$.

2.4.1.2.2 Constructing the Hypothesis Test In the previous section we described a procedure for creating a fixed-width once-sided sequential upper confidence limit with coverage probability $1-\alpha$. In this section, we describe the process to convert the one-sided upper confidence interval to a level-α hypothesis test using the duality of confidence intervals and hypothesis tests.

Lemma 2.4.1. *If $\hat{s}+w$ be an upper confidence limit for s with coverage probability $1-\alpha$, then a level $-\alpha$ hypothesis test for null hypothesis $H0: s \geq t$ against alternate hypothesis $H1: s < t$ will be to Reject H0, if $\hat{s}+w < t$, else Fail to Reject H0.*

Proof. By equation 2.5,

$$P(\hat{S}+w \geq s) = 1-\alpha$$

$$\implies P(\hat{S}+w \geq t | s \geq t) \geq 1-\alpha$$

$$\implies -P(\hat{S}+w \geq t | s \geq t) \leq -1+\alpha$$

$$\implies 1-P(\hat{S}+w \geq t | s \geq t) \leq \alpha$$

$$\implies P(\hat{S}+w < t | s \geq t) \leq \alpha$$

$$\implies P(Reject\ H0 | H0) \leq \alpha$$

□

2.4.1.2.3 Choosing w The fixed-width w of the one-sided upper confidence interval has a significant effect on the efficiency of the test. Intuitively, the larger the width w, the less the number of hash comparisons required to attain a confidence interval of that width. However, setting w to a very high value would result in a large Type II error for our test. Though our algorithm's quality is not affected by Type II error (since we compute exact or approximate

similarity when alternate hypothesis is satisfied), but still a large Type II error will imply that many false positives (candidates which fall in alternate hypothesis, but are classified as null hypothesis). Exact/approximate similarity is computed and these candidates are pruned away. Therefore, a large Type II error will translate to lower efficiency. In other words, making w too high or too low will cause significant slowdown in terms of execution time.

We next describe a simple heuristic to select w. Suppose a candidate pair has similarity $s < t$, i.e. for this candidate pair, the null hypothesis should be rejected. So the upper confidence limit $\hat{s} + w$ can be as high as t, and our test statistic should still be able to reject it. Therefore, the maximum length of w is dictated by how large the upper confidence limit can be. So instead of presetting w to a fixed value, we dynamically set w according to the following heuristic. We compare the first batch of hashes and use the crude estimate of s, say $\hat{si}$ from the first batch to come up with w:

$$w = t - \hat{si} - \varepsilon \tag{2.8}$$

The key insight here is, *instead of using a single hypothesis test for all candidate pairs, we choose a different hypothesis test based on an initial crude similarity estimate of the candidate pair being analyzed, so that w can be maximized, while still keeping Type II error in control, resulting in efficient pruning.* Note that every such test is a $level - \alpha$ test according to Lemma 2.4.1. We need the ε parameter as $\hat{si}$ is a crude estimate from the first batch of hash comparisons and it could be an underestimate, which would result in an overestimate of w. Consequently, the final test statistic $\hat{s} + w$ can go beyond t and the candidate cannot be pruned. Figure 2.1 explains the phenomenon. Of course, dynamically constructing the test for each candidate can make the candidate pruning process inefficient. We solve this issue by caching a number of tests for different w and during the candidate

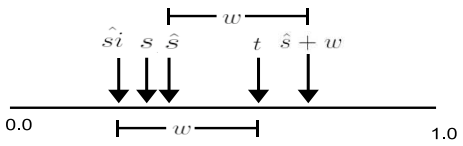


Figure 2.1: Estimating w

pruning step, the test that is closest to *w* (but smaller than or equal to it) is selected. Hence there is no need for online inference, making the algorithm very efficient.

2.4.1.3 Hybrid Hypothesis Tests

We found out empirically that the number of hash comparisons required by SPRT is very high for our composite hypothesis test problem. The one-sided CI based tests performed considerably better. Specifically we saw that the candidate pairs whose actual similarity *s* is quite far away from the threshold *t* were very quickly classified into the null or alternate hypothesis as the width *w* is quite large. But interestingly, for the candidate pairs which have similarity very close to the threshold, the estimated parameter *w* becomes very small. For such pairs, to attain the fixed-width confidence interval, the number of hash comparison requirement is very high. It is even higher than the more traditional SPRT. Therefore, to utilize the best of both worlds, we use a hybrid hypothesis testing strategy, where based on how far the true similarity is away from the threshold, we either select a one-sided CI based hypothesis test, or the SPRT. Formally, *we use a parameter* μ*, such that if the estimated fixed-width* $w \geq \mu$*, we use the one-sided CI based hypothesis test, else we use SPRT*. Again, in this hybrid strategy all the tests are $level - \alpha$, so we have guarantees on the overall Type I error, while minimizing the number of overall hash comparisons by smarty selecting the proper test for a specific candidate.

2.4.2 Concentration Inference

To solve the concentration inference of equation 2.4, we can create a two-sided fixed-width confidence interval for a binomial proportion under the sequential setting. The technique is very similar to the one we described in section 2.4.1.2 for the one-sided upper confidence limit. The major difference is, for two-sided confidence interval, the coverage probability equation 2.6 from section 2.4.1.2 becomes:

$$T(s,\lambda) = \sum_{i=1}^{k} H(m_i,n_i)s^{m_i}(1-s)^{n_i-m_i}I(|s-\frac{m_i}{n_i}| \leq \delta)$$

Now this equation can be solved in a manner similar to the one described in section 2.4.1.2 to find out the critical value λ and hence the stopping points in the sequential process. The stopping rule will also change to $z_{\frac{\lambda}{2}}\sqrt{\frac{\hat{s_a}(1-\hat{s_a})}{n}} \leq \delta$ (in the one-sided case λ was used instead of $\frac{\lambda}{2}$). The concentration inference is used to estimate the similarity with probabilistic guarantees under circumstances where exact similarity computation is infeasible.

Choosing maximum number of hashes: In our problem scenario, we do not need all candidates to converge to a fixed-width interval. Since we guarantee $1-\alpha$ recall, any candidate pair which has less than α probability of being greater than t can be ignored. In other words, we do not need to consider all the stopping points generated. We can choose the stopping points based on the user defined threshold t.

Lemma 2.4.2. *If m_i, n_i are the stopping points decided by the fixed-width confidence interval method having coverage probability $1-\gamma$, the stopping points m_i, n_i, such that $\frac{m_i}{n_i} < t-\delta$ can occur with probability at most γ if the true similarity s is greater than threshold t.*

Proof. By the fixed-width confidence interval guarantee,

$$P(\hat{S} - \delta \leq s \leq \hat{S} + \delta) = 1 - \gamma$$

$$\implies P(s > \hat{S} + \delta \text{ or } s < \hat{S} - \delta) = \gamma$$

$$\implies P(\hat{S} < s - \delta) \leq \gamma$$

This implies for $s > t$, the probability of $\hat{s} < t - \delta$ is less than γ. For all the stopping points m_i, n_i, $\hat{s} = \frac{m_i}{n_i}$. Hence the proof. □

Corollary 2.4.3. *If m_i, n_i are the set of stopping points, the maximum number of hashes n_{max} required by our algorithm to estimate any similarity above t while ensuring $1 - \gamma$ recall is $n_{max} = max(n_i)$ s.t. $\frac{m_i}{n_i} \geq t - \delta$.*

If we set $\gamma = \alpha$, the above lemma will ignore points which have less than α probability of being a true positive. In other words, our algorithm is able to guarantee $1 - \alpha$ recall.

2.4.3 Similarity Measures

The proposed techniques in this chapter can be used with similarity measures for which a locality sensitive hash function exists. Previous work has developed locality sensitive hash functions for a wide range of similarity measures [25, 26, 45, 73, 79, 127] as well as for arbitrary kernel functions [92]. In this chapter, we show that our methods work well with two of the most popular similarity measures - i) Jaccard similarity and ii) Cosine similarity.

2.4.3.1 Jaccard Similarity

The locality sensitive hash function relevant to Jaccard similarity is *MinWise Independent Permutation*, developed by Broder *et al* [25]. This hash function can approximate the Jaccard coefficient between two sets x, y. Formally,

$$P(h(x) == h(y)) = \frac{|x \cap y|}{|x \cup y|}$$

The estimate of Jaccard similarity between x, y will be:

$$\hat{s} = \frac{1}{n}\sum_{i=1}^{n} I[h_i(x) == h_i(y))]$$

As described earlier in equation 2.2, the likelihood function for getting m matches out n hashes is a binomial with parameters n, s. Note that n is also a random variable here. Hence we can directly use our proposed methods for doing inference on s.

2.4.3.2 Cosine Similarity

The locality sensitive hash function for cosine similarity is given by the *rounding hyperplane algorithm*, developed by Charikar [36]. However, the similarity given by the above algorithm is a little different from cosine similarity. Specifically, such a hash function gives:

$$P(h(x) == h(y)) = 1 - \frac{\theta}{\pi}$$

where θ is the angle between the two vectors. Let the above similarity be defined as s and let the cosine similarity be r. The range of s is therefore, 0.5 to 1.0. To convert between s and r, we need the following transformations:

$$r = cos(\pi(1-s)) \tag{2.9}$$

$$s = 1 - \frac{cos^{-1}(r)}{\pi} \tag{2.10}$$

Consequently, we need to adapt our proposed algorithms to handle these transformations. Handling the pruning inference is quite simple. If the user sets the cosine similarity threshold as t, before running our pruning inference, we change the threshold to the value of the transformed similarity measure. So the pruning inference becomes $s \geq (1 - \frac{cos^{-1}(t)}{\pi})$ instead of $s \geq t$.

The transformation of the concentration inference is trickier. We need to transform the confidence interval of s (our algorithm will generate this) to the confidence interval of r (for cosine similarity). The user provides an estimation error bound δ, implying that we need to generate an estimate $\hat{r}$ within a confidence interval of 2δ with $1-\gamma$ coverage probability. Since we can only estimate $\hat{s}$, we need to create a level-$(1-\gamma)$ confidence interval $2\delta_s$ around $\hat{s}$, such that, if $l_s \leq s \leq u_s$ and $l_r \leq r \leq u_r$ then,

$$u_s - l_s \leq 2\delta_s \implies u_r - l_r \leq 2\delta$$

If we create, a $2\delta_s$ fixed-width confidence interval, then the upper and lower confidence limits will be $\hat{s}+\delta_s$ and $\hat{s}-\delta_s$ respectively. Since r is a monotonically increasing function of s, hence the upper and lower confidence limit of r (cosine similarity) will be $cos(\pi(1-min(1.0,\hat{s}+\delta_s)))$ and $cos(\pi(1-max(0.5,\hat{s}-\delta_s)))$ respectively. The interval for the estimate of cosine similarity will be $cos(\pi(1-min(1.0,\hat{s}+\delta_s))) - cos(\pi(1-max(0.5,\hat{s}-\delta_s)))$. We have to choose δ_s such that

$$cos(\pi(1-min(1.0,\hat{s}+\delta_s))) - cos(\pi(1-max(0.5,\hat{s}-\delta_s))) \leq 2\delta$$

The above function is monotonically decreasing in $\hat{s}$. So the interval will be largest when $\hat{s}$ is smallest (0.5). So we set $\hat{s}$ to 0.5 and numerically find out largest δ_s such that the inequality of the above expression is still satisfied.

2.5 Experimental Evaluation

2.5.1 Experimental Setup and Datasets

We ran our experiments on a single core of an AMD Opteron 2378 machine with 2.4GHz CPU speed and 32GB of RAM. We only use one core as our application is a single threaded

Dataset	Vectors	Dimensions	Avg. Len	Nnz
Twitter	146,170	146,170	1369	200e6
WikiWords100K	100,528	344,352	786	79e6
RCV	804,414	47,236	76	61e6
WikiLinks	1,815,914	1,815,914	24	44e6
Orkut	3,072,626	3,072,66	76	233e6

Table 2.1: Dataset details

C++ program. We use five real world datasets to evaluate the quality and efficiency of our all pairs similarity search algorithms. Details are given in Table 2.1

Twitter: This is a graph representing follower-followee links in Twitter [93]. Only users having at least 1000 followers are selected. Each user is represented as an adjacency list of the users it follows.

WikiWords100K and WikiLinks: These datasets are derived from the English Wikipedia Sep 2010 version. The WikiWords100K is a preprocessed text corpus with each article containing at least 500 words. The Wikilinks is a graph created from the hyperlinks of the entire set of articles weighted by tf-idf.

RCV: This is a text dataset consisting of Reuters articles [98]. Each document is represented as a set of words. Some basic preprocessing such as stop-words removal and stemming is done.

Orkut: The Orkut dataset is a friendship graph of 3 million users weighted by tf-idf [112].

2.5.2 Results

As explained in section 2.4 (methodology), we expect Bayes-LSH/Lite variants to be very fast, however, those could potentially suffer a loss in the qualitative guarantees as they model

an inherently sequential process in a non-sequential manner. Since the sequential confidence interval based methods have provable guarantees about quality (lemmas 2.4.1 and 2.4.2), they are always expected to be qualitatively better. The SPRT and hence the Hybrid methods should qualitatively perform very well, however under the composite hypothesis testing scenario, SPRT cannot provide strong guarantees. In the next two sections, we will evaluate these premises.

2.5.2.1 Algorithms using Early Pruning and Exact Similarity Computation

We compare the following four strategies for computing all pairs with similarity above a certain user defined threshold. All of these algorithms assume that the original dataset is available (instead of the smaller sketch). These algorithms use the exact candidate generation technique AllPairs [18] and an early pruning technique and finally exact similarity computation.

BayesLSHLite: BayesLSHLite [131] is the state-of-the-art candidate pruning algorithm which is known to perform better than AllPairs [18] and PPJoin [161].

SPRT: We use the traditional Sequential Probability Ratio Test to do the early pruning of candidates. We set $\tau = 0.025$.

One-Sided-CI-HT: We compare against our model, which is the fixed-width one-sided upper confidence interval based hypothesis testing technique. We set $\varepsilon = 0.01$. The choice of ε is done by empirically evaluating several values – we found values in the neighborhood of $0.01 - 0.05$ worked best. We set $a = 4$ as it seems to work well in practice [61].

Hybrid-HT: This is the second model we propose, where based on the candidate in question, we either choose a One-Sided-CI-HT or SPRT. We set $\mu = 0.18$, that is the threshold of w below which our Hybrid-HT algorithm switches from a One-Sided-CI-HT to SPRT. Again, we selected μ, empirically by trying different thresholds. For all the tests above, we set

the Type I error or recall parameter $1-\alpha = 0.97$. We also compare with the AllPairs algorithm which uses exact similarity computation right after the candidate generation step (no candidate pruning).

The performance and quality numbers are reported in Figure 2.2. We measure performance by total execution time and we measure quality by recall (since we are giving probabilistic guarantees on recall). An added advantage of these methods is that since we compute exact similarity for all candidates that are retained and check whether they are above the threshold using exact similarity computation, all the strategies yield full precision (100%). Furthermore, the sequential hypothesis tests we do are truncated tests, i.e. we compute at most $h = 256$ hashes, after which if a decision cannot be made, we send the pair for exact similarity computation. We report results on all the aforementioned datasets on both Jaccard and cosine similarity measures. For Jaccard, we vary the similarity threshold from $0.3-0.7$ and for cosine, we vary the threshold from $0.5-0.9$. These are the same parametric settings used in the original BayesLSHLite work.

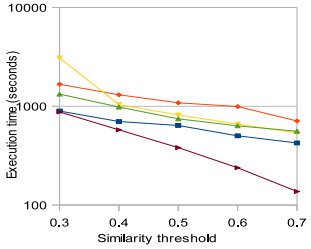
Results indicate that the pattern is quite similar for all datasets. BayesLSHLite is always substantially faster in the case of cosine similarity while in the case of Jaccard similarity, AllPairs is marginally faster at times. At high values of the similarity threshold, SPRT is the slowest, while both One-Sided-CI-HT and Hybrid-HT performs very close to BayesLSHLite. This performance benefit comes from the one-sided tests. More precisely, choosing the width w of the test based on the estimate from the first bin of hash comparisons makes each test optimized for the specific candidate being processed. Those tests are extremely efficient at pruning away false positive candidates whose true similarities are very far from the similarity threshold t. The reason is these tests can allow a larger confidence width w and hence less number of hash comparisons. Obviously, the Hybrid-HT performs very well

at such high thresholds because it chooses one of the one-sided tests. However, at the other end of the spectrum, at very low similarity thresholds, the allowable confidence interval width w becomes too small and a large number of trials is required by the one-sided tests, making them inefficient. SPRT performs reasonably well under these situations. Under these conditions, the Hybrid-HT strategy is able to perform better than both (SPRT and One-Sided-CI-HT) as it able to smartly delegate pairs with true similarity close to the threshold to SPRT instead of one-sided tests. In summary, the green lines (Hybrid-HT) can perform well through the whole similarity threshold range. For the WikiWords100K dataset in Figure 2.2k Hybrid-HT gave 8.8x speedup over AllPairs and 2.1x speedup of SPRT at 0.9 threshold and at 0.5 threshold, it gave 3.4x speedup over AllPairs and a 1.3x speedup over SPRT.

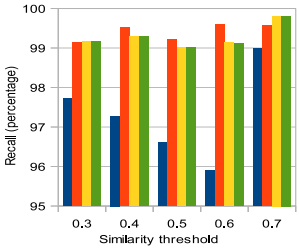
In terms of quality, our proposed method One-Sided-CI-HT guarantee at least 97% recall ($\alpha = 0.03$). In all results, we see the recall of One-Sided-CI-HT, as expected, is above 97%. In spite of the fact that SPRT does not have strong guarantees in case of composite hypothesis, we see that SPRT performs quite well in all datasets. Since Hybrid-HT uses the One-Sided-CI-HT and SPRT, its quality numbers are also extremely good. Only BayesLSHLite, which does not model the hash comparisons as a sequential process falls marginally below the 97% mark at some places. In summary, our tests can provide rigorous quality guarantees, while significantly improving the performance by over traditional SPRT.

2.5.2.2 Algorithms using Early Pruning and Approximate Similarity Estimation

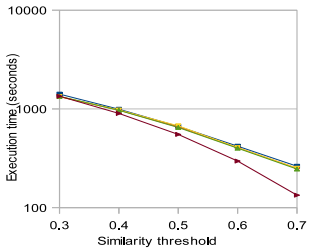
The previous section discussed the algorithms that can be used when the explicit representation of the original data is available. We now describe results on two algorithms for which only the hash signatures needs to be stored rather than the entire dataset. These



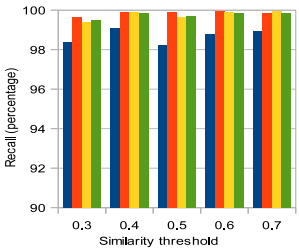
(a) Twitter, Jaccard



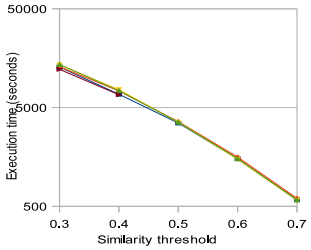
(b) Twitter, Jaccard



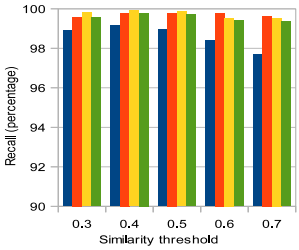
(c) WikiWords100K,Jaccard



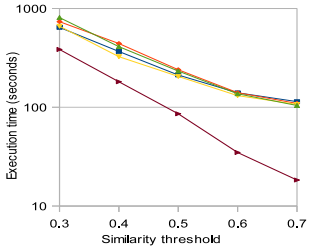
(d) WikiWords100K,Jaccard



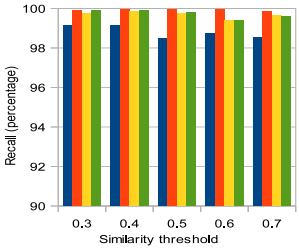
(e) RCV,Jaccard



(f) RCV,Jaccard



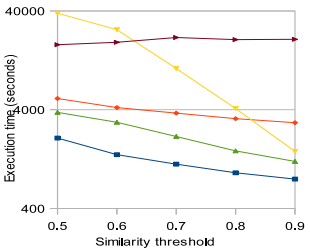
(g) WikiLinks,Jaccard



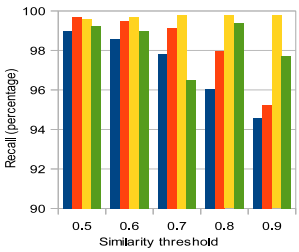
(h) WikiLinks,Jaccard

Figure 2.2: Comparisons of algorithms with exact similarity computation (CONTINUED).

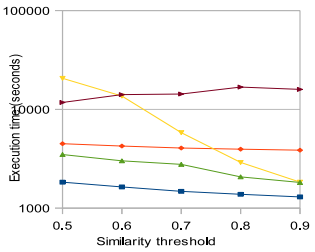
Figure 2.2: CONTINUED



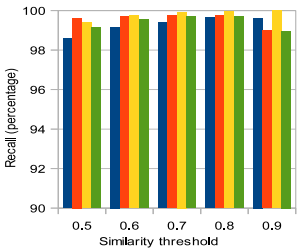
(i) Twitter,cosine



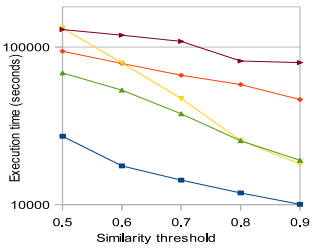
(j) Twitter,cosine



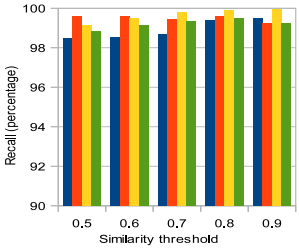
(k) WikiWords100K,cosine



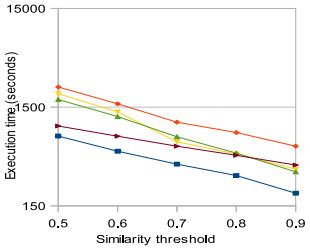
(l) WikiWords100K,cosine



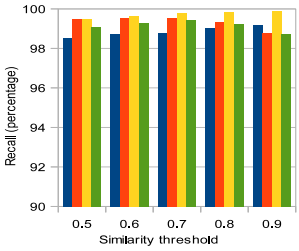
(m) RCV,cosine



(n) RCV,cosine



(o) WikiLinks,cosine



(p) WikiLinks,cosine

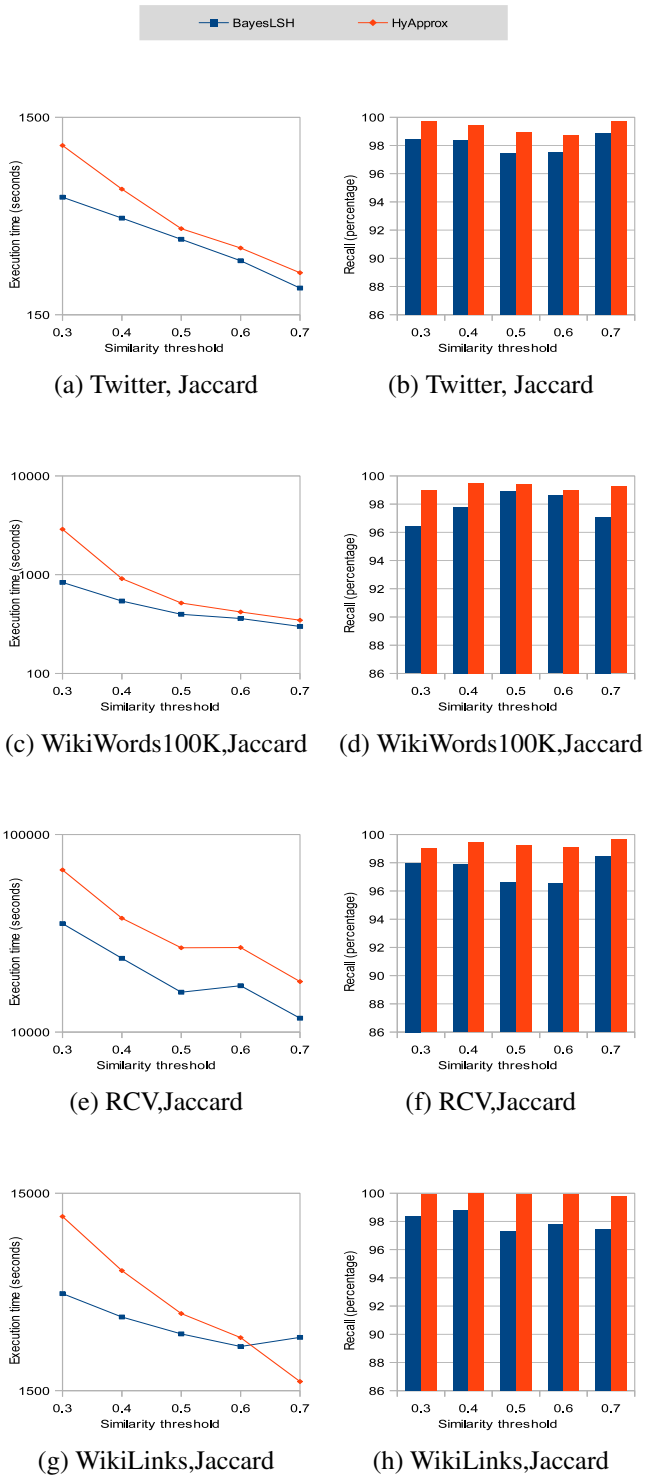
algorithms use the LSH index generation followed by candidate pruning, followed by approximate similarity estimation. We compare the following two techniques:

BayesLSH: This uses the same pruning technique as Bayes-LSHLite along with the concentration inference for similarity estimation.

Hybrid-HT-Approx: This is our sequential variant. It uses Hybrid-HT's pruning technique along with the sequential fixed-width confidence interval generation strategy as described in section 2.4.2. We set $\tau = 0.015$.

We use the same parametric settings as before. The additional parameters required here are the estimation error bound δ and the coverage probability $1-\gamma$ for the confidence interval. We set $\delta = 0.05$ and $\gamma = \alpha$. Again we measure performance by execution time. Here we measure quality by both recall and estimation error as we provide probabilistic guarantees on both.

Figure 2.3 reports both the performance and recall numbers. We do not list the estimation error numbers as the avg. estimation error for each algorithm on each dataset was within the specified bound of 0.05. Results indicate that Hybrid-HT-Approx is slower than BayesLSH as expected, however, is qualitatively better than BayesLSH. More importantly, in all cases, Hybrid-HT-Approx has a recall value which is well above the 97% guaranteed number. BayesLSH on an average performs quite well, however it does fall below the guaranteed recall value quite a few times. In summary, our method provides rigorous guarantees of quality without losing too much performance over BayesLSH.



(a) Twitter, Jaccard (b) Twitter, Jaccard

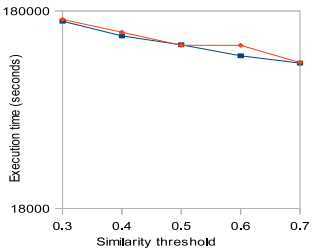
(c) WikiWords100K,Jaccard (d) WikiWords100K,Jaccard

(e) RCV,Jaccard (f) RCV,Jaccard

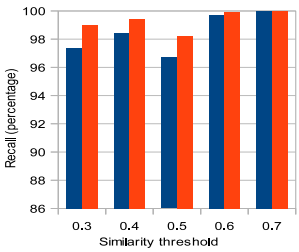
(g) WikiLinks,Jaccard (h) WikiLinks,Jaccard

Figure 2.3: Comparisons of algorithms with approximate similarity estimations (CONTINUED).

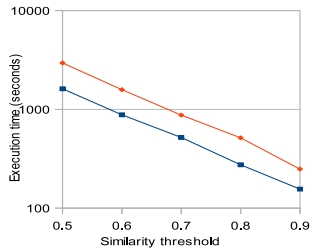
Figure 2.3: CONTINUED



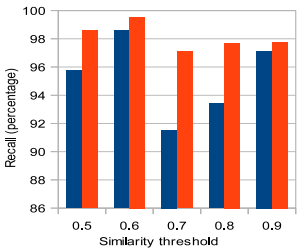
(i) Orkut,Jaccard



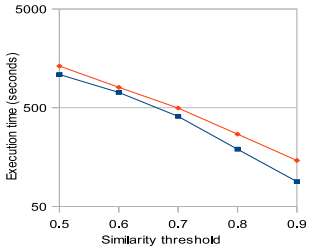
(j) Orkut,Jaccard



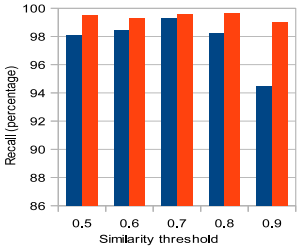
(k) Twitter, cosine



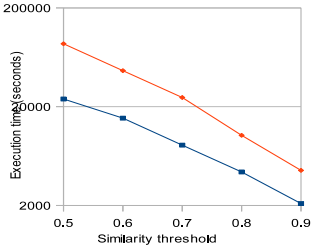
(l) Twitter, cosine



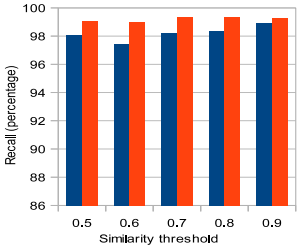
(m) WikiWords100K,cosine



(n) WikiWords100K,cosine



(o) RCV,cosine



(p) RCV,cosine

2.6 Conclusions

In this chapter, we propose principled approaches of doing all pairs similarity search on a database of objects with a given similarity measure. We describe algorithms for handling two different scenarios - i) the original data set is available and the similarity of interest can be exactly computed from the explicit representation of the data points and ii) instead of the original dataset only a small sketch of the data is available and similarity needs to be approximately estimated. For both scenarios we use LSH sketches (specific to the similarity measure) of the data points. For the first case we develop a fully principled approach of adaptively comparing the hash sketches of a pair of points and do composite hypothesis testing where the hypotheses are similarity greater than or less than a threshold. *Our key insight is a single test does not perform well for all similarity values, hence we dynamically choose a test for a candidate pair, based on a crude estimate of the similarity of the pair*. For the second case, we additionally develop an adaptive algorithm for estimating the approximate similarity between the pair. Our methods are based on finding sequential fixed-width confidence intervals. We compare our methods against state-of-the-art all pairs similarity search algorithms – BayesLSH/Lite that does not precisely model the adaptive nature of the problem. We also compare against the more traditional sequential hypothesis testing technique – SPRT. We conclude that if the quality guarantee is paramount, then we need to use our sequential confidence interval based techniques, and if performance is extremely important, then BayesLSH/Lite is the obvious choice. Our Hybrid models give a very good tradeoff between the two extremes. We show that our hybrid methods always guarantee the minimum prescribed quality requirement (as specified by the input parameters) while being up to 2.1x faster than SPRT and 8.8x faster than AllPairs. Our

hybrid methods also improve the recall by up to 5% over BayesLSH/Lite, a contemporary state-of-the-art adaptive LSH approach.

Chapter 3: Generalizing Locality Sensitive Hashing Based Similarity Estimation to Kernels

The previous chapter described an approximate solution with rigorous quality guarantees to the APSS task. This was done in the context of vanilla similarity measures such as Jaccard index and cosine similarity. However, to generalize the usage of LSH for similarity estimation based application across complex application domains, it is paramount to support LSH for a general class of similarity measure. In this chapter, we develop an *unbiased LSH estimator* for similarity search for the class of kernel similarity measures (capable of representing inner product similarities).

3.1 Introduction

In recent past, Locality Sensitive Hashing (LSH) [9] has gained widespread importance in the area of large scale machine learning. Given a high dimensional dataset and a distance/similarity metric, LSH can create a small sketch (low dimensional embedding) of the data points such that the distance/similarity is preserved. LSH is known to provide an approximate and efficient solution for estimating the pairwise similarity among data points, which is critical in solving applications for many domains ranging from image retrieval to text analytics and from protein sequence clustering to pharmacogenomics. Recently kernel-based similarity measures [137] have found increased use in such scenarios in part because

the data becomes easily separable in the kernel induced feature space. The challenges of working with kernels are twofold – (1) explicit embedding of data points in the kernel induced feature space (RKHS) may be unknown or infinite dimensional and (2) generally, the kernel function is computationally expensive. The first problem prohibits the building of a smart index structure such as kdtrees [20] that can allow efficient querying, while the second problem makes constructing the full kernel matrix infeasible.

LSH has been used in the context of kernels to address both of the aforementioned problems. Existing LSH methods for kernels [81, 92] leverage the KPCA or Nyström techniques to estimate the kernel. The two methods differ only in the form of covariance operator that they use in the eigenvector computation step to approximately embed the data in RKHS. While KPCA uses the centered covariance operator, Nyström method uses the uncentered one (second moment operator). Without loss of generality, for the rest of the chapter, we will use the Nyström method and hence by covariance operator, we will mean the uncentered one. The LSH estimates for kernel differ significantly from the Nyström approximation. This is due to the fact that the projection onto the subspace (spanned by the eigenvectors of covariance operator) results in a reduction of norms of the data points. This reduction depends on the eigenvalue decay rate of the covariance operator. Therefore, this norm reduction is difficult to estimate apriori. Assume that the original kernel was normalized with norm of the data points (self inner product) equaling 1. As a consequence of this norm reduction, in the resulting subspace the Nyström approximated kernel is not normalized (self inner product less than 1). Now, it is shown in [36] that LSH can only estimate normalized kernels. Thus in the current setting, instead of the Nyström approximated kernel, it estimates the *re-normalized version of it*. The bias arising out of this re-normalization depends on the eigenvalue decay rate of the covariance operator and is

unknown to the user apriori. This is particularly problematic since for the LSH applications (index building and estimation) in the context of similarity (not distance), accurate estimation is paramount. For instance, the *All Pairs Similarity Search* (APSS) [18, 30, 33] problem finds all pairs of data points whose similarity is above a user defined threshold. Therefore, APSS quality will degrade in the case of high estimation error. In APSS using LSH [33], it is clearly noticeable that the quality of non-kernel similarity measures is better than their kernel counterparts.

We propose a novel embedding of data points that is amenable to LSH sketch generation, while still estimating the Nyström approximated kernel matrix instead of the re-normalized version (which is the shortcoming of existing work). Specifically, the contributions of this chapter are as follows:

1. We show that Nyström embedding based LSH generates the LSH embedding for a slightly different kernel rather than the Nyström approximated one. This bias becomes particularly important during the LSH index construction where similarity threshold (or distance radius) is a mandatory parameter. Since this radius parameter is given in terms of the original similarity (kernel) measure, if the LSH embedding results in a bias (estimating a slightly different kernel), then the resulting index generated will be incorrect.

2. We propose an LSH scheme to estimate the Nyström approximation of the original input kernel and develop an algorithm for efficiently generating the LSH embedding.

3. Finally, we empirically evaluate our methods against state-of-the-art KLSH [81, 92] and show that our method is substantially better in estimating the original kernel values. We additionally run statistical tests to prove that the statistical distribution

of pairwise similarity in the dataset is better preserved by our method. Preserving the similarity distribution correctly is particularly important in applications such as clustering.

Our results indicate up to 9.7x improvement in the kernel estimation error and the KL divergence and Kolmogorov-Smirnov tests [105] show that the estimates from our method fit the pairwise similarity distribution of the ground truth substantially better than the state-of-the-art KLSH method.

3.2 Background and Related Works

3.2.1 LSH for Cosine Similarity

A family of hash functions F is said to be locality sensitive with respect to some similarity measure, if it satisfies the following property [36]:

$$P_{h\in F}(h(x) = h(y)) = sim(x,y) \tag{3.1}$$

Here x, y is a pair of data points, h is a hash function and sim is a similarity measure of interest. LSH for similarity measures can be used in two ways:

1. **Similarity Estimation:** If we have k i.i.d. hash functions $\{h_i\}_{i=1}^{k}$, then a maximum likelihood estimator (MLE) for the similarity is:

$$\widehat{sim(x,y)} = \frac{1}{k}\sum_{i=1}^{k} I(h_i(x) = h_i(y)) \tag{3.2}$$

2. **LSH Index Search:** The concatenation of the aforementioned k hash functions form a signature and suppose l such signatures are generated for each data point. Then for a query data point q, to find the nearest neighbor, only those points that have at least one signature in common with q need to be searched. This leads to an index construction

n, d	Number of data points, Dimensionality of data
p, c	Parameters: Number of eigenvectors to use, Number of extra dimensions to use
$\kappa(x, y)$	Kernel function over data points x, y ($=< \Phi(x), \Phi(y) >$)
$\Phi(x)$	Kernel induced feature map for data point x
X_i	i^{th} data point (i^{th} row of matrix $X_{n\times d}$)
$K_{i,j}$	$(i,j)^{th}$ value of true kernel matrix $K_{n\times n}$ ($= \kappa(X_i, X_j)$)
Y_i	p dimensional Nystroḿ embedding of $\Phi(X_i)$ (i^{th} row of $Y_{n\times p}$ matrix)
$\widehat{K_{i,j}}$	Approximation of $K_{i,j}$ due to Nystroöm embedding ($=< Y_i, Y_j >$)
Z_i	Our $(p+n)$ dimensional Augmented Nystroḿ embedding of $\Phi(X_i)$
$\widehat{K_{(Z)i,j}}$	Approximation of $K_{i,j}$ due to Augmented Nystroḿ embedding ($=< Z_i, Z_j >$)
Z'_i	Our $(p+c)$ dimensional Remapped Augmented Nystroḿ embedding of $\Phi(X_i)$

Table 3.1: Key Symbols

algorithm that results in a sublinear time search. It is worth noting that a similarity threshold is a mandatory parameter for an LSH index construction. Consequently, a bias in its estimation may lead to a different index than the one intended based on input similarity measure.

Charikar [36] introduced a hash family based on the rounding hyperplane algorithm that can very closely approximate the cosine similarity. Let $h_i(x) = sign(r_i x^T)$, where $r_i, x \in R^d$ and each element of r_i is drawn from i.i.d. $N(0,1)$. Essentially the hash functions are signed random projections (SRP). It can be shown that in this case,

$$P(h_i(x) = h_i(y)) = 1 - \frac{\theta(x,y)}{\pi} = sim(x,y)$$

$$\implies cos(\theta(x,y)) = cos(\pi(1 - sim(x,y)))$$

where $\theta(x,y)$ is the angle between x,y. The goal of this work is to find a locality sensitive hash family for the Nyström approximation $\hat{\kappa}$ of any arbitrary kernel κ that will satisfy the following property:

$$P(h_i(x) = h_i(y)) = 1 - \frac{cos^{-1}(\widehat{\kappa(x,y)})}{\pi} \tag{3.3}$$

3.2.2 Existence of LSH for Arbitrary Kernels

Kernel similarity measures are essentially the inner product in some transformed feature space. The transformation of the original data into the kernel induced feature space is usually non-linear and often explicit embedding in the kernel induced space are unknown, only the kernel function can be computed. Shrivastava *et. al.* [139] recently proved the non-existence of LSH functions for general inner product measures. In spite of the non-existence of LSH for kernels in the general case, LSH can still exist for a special case, where the kernel is normalized – in other words the inner product is equal to the cosine similarity measure. As mentioned in previous section, Charikar [36] showed that using signed random projections, cosine similarity can be well approximated using LSH. To summarize, LSH in kernel context is meaningful in the following two cases:

1. The case where the kernel is normalized with each data object in the kernel induced feature space having unit norm.

$$||\Phi(x)||^2 = \kappa(x,x) = 1 \tag{3.4}$$

   Here $\kappa(.,.)$ is the kernel function and $\Phi(.)$ is the (possibly unknown) kernel induced feature map in RKHS.

2. In the case equation 3.4 does not hold, LSH does not exist for $\kappa(.,.)$. But it exists for a normalized version of κ, say $\kappa_N(.,.)$, where:

$$\kappa_N(x,y) = \frac{\kappa(x,y)}{\sqrt{\kappa(x,x)}\sqrt{\kappa(y,y)}} \tag{3.5}$$

3.2.3 Kernelized Locality Sensitive Hashing

KLSH [92] is an early attempt to build an LSH index for any arbitrary kernel similarity measure. Later work by Xia *et. al.* [160] tries to provide bounds on kernel estimation error using Nyström approximation [159]. This work also provides an evaluation of applying LSH directly on explicit embedding generated by KPCA [133]. A follow up [81] to KLSH provided further theoretical insights into KLSH retrieval performance and proved equivalence of KLSH and KPCA+LSH.

KLSH computes the dot product of a data point and a random Gaussian in the approximate RKHS spanned by the first p principal components of the empirically centered covariance operator. It uses an approach similar to KPCA to find out a data point's projection onto the eigenvectors in the kernel induced feature space and it approximates the random Gaussian in the same space by virtue of the central limit theorem (CLT) of Hilbert spaces by using a sample of columns of the input kernel matrix. Let $X_{n\times d}$ denote the dataset of n points, each having d dimensions. We denote the i^{th} row/data point by X_i and i,j^{th} element of X by $X_{i,j}$. Let $K_{n\times n}$ be the full kernel matrix ($K_{i,j} = \kappa(X_i, X_j)$). KLSH takes as input p randomly selected columns from kernel matrix - $K_{n\times p}$. The algorithm to compute the hash bits is as follows:

1. *Extract $K_{p\times p}$ from input $K_{n\times p}$. $K_{p\times p}$ is a submatrix of $K_{n\times n}$ created by sampling the same p rows and columns.*

2. *Center the matrix* $K_{p\times p}$.

3. *Compute a hash function h by forming a binary vector e by selecting t indices at random from* $1,...,p$, *then form* $w = K_{p\times p}^{-1/2} e$ *and assign bits according to the hash function*

$$h(\Phi(X_a)) = sign(\sum_i w(i)\kappa(X_i, X_a))$$

One thing worth noting here is, unlike vanilla LSH, where an LSH estimator tries to estimate the similarity measure of interest directly, in case of KLSH, the estimator tries to estimate the kernel similarity that is approximated by the KPCA embedding. The idea is that the KPCA embedding should lead to good approximations of the original kernel and hence KLSH should be able to approximate the original kernel as well. Alternatively, instead of directly computing the dot product in RKHS, one may first explicitly compute the KPCA/Nyström $p-$dimensional embedding of the input data and generate a $p-$dimensional multivariate Gaussian, and then compute the dot product. The two methods are equivalent [81]. Next, we discuss why approximation error due to applying LSH on kernels may be significant.

3.3 Estimation Error of LSH for Kernels

According to Mercer's theorem [109], the kernel induced feature map $\Phi(x)$ can be written as $\Phi(x) = [\phi_i(x)]_{i=1}^{\infty}$ where $\phi_i(x) = \sqrt{\sigma_i}\psi_i(x)$ and σ_i and ψ_i are the eigenvalues and eigenfunctions of the covariance operator whose kernel is κ. The aforementioned infinite dimensional kernel induced feature map can be approximated explicitly in finite dimensions by using Nyström style projection [159] as described next. This can be written as $\widehat{\Phi(x)} = [\widehat{\phi_i(x)}]_{i=1}^{p}$ where $\widehat{\phi_i(x)} = \frac{1}{\sqrt{\lambda_i}} < K(x,.), u_i >$. Here $K(x,.)$ is a vector containing

the kernel values of data point x to the p chosen points, λ_i and u_i are the i^{th} eigenvalue and eigenvector of the sampled $p \times p$ kernel matrix $K_{p\times p}$. Note that, both the KPCA and Nyström projections are equivalent other than the fact that in the case of KPCA, $K_{p\times p}$ is centered, whereas in the case of Nyström, it is uncentered. Essentially, $\widehat{\Phi(x)} = P_{\hat{S}}\Phi(x)$, where $P_{\hat{S}}$ is the projection operator that projects $\Phi(x)$ onto the subspace spanned by first p eigenvectors of the empirical covariance operator. Let $Y_{n\times p}$ represent this explicit embedding of the data points.

In the next lemma, we show that the above approach results in a bias for kernel similarity approximation from LSH.

Lemma 3.3.1. *If $\widehat{K_{(LSH)i,j}}$ is the quantity estimated by using LSH on Nyström embedding, then $\widehat{K_{(LSH)i,j}} \geq \widehat{K_{i,j}}$.*

Proof. Since $\widehat{K_{(LSH)}}$ is the quantity estimated by the LSH estimator for cosine similarity on embedding $Y_{n\times p}$, then by equation 3.5

$$\widehat{K_{(LSH)i,j}} = \frac{Y_i Y_j^T}{||Y_i||||Y_j||} = \frac{\widehat{K_{i,j}}}{\sqrt{\widehat{K_{i,i}}}\sqrt{\widehat{K_{j,j}}}} \tag{3.6}$$

where Y_i is the i^{th} row of Y.

By assumption, $||\Phi(X_i)|| = 1,\ \forall i$. Hence

$$\widehat{K_{i,i}} = < P_{\hat{S}}\Phi(X_i), P_{\hat{S}}\Phi(X_i) > = ||P_{\hat{S}}\Phi(X_i)||^2 \leq 1,\ \forall i$$

(since $P_{\hat{S}}$ is a projection operator onto a subspace). Specifically if $i \in p$, then $\widehat{K_{i,i}} = K_{i,i}$. Putting $\widehat{K_{i,i}} \leq 1$ in equation 3.6, we get the following.

$$\widehat{K_{(LSH)i,j}} \geq \widehat{K_{i,j}}$$

□

Thus, applying LSH to the Nyström embedding results in an overestimation of the kernel similarity when compared to the Nyström approximation to the kernel similarity. In terms of our goal, equation 3.3 will have $\widehat{K_{(LSH)}}$ instead of $\hat{K}$ (Nyström approximated kernel). Unlike $\hat{K}$, $\widehat{K_{(LSH)}}$ does not approximate K (true kernel) well, unless p is extremely large. This is not feasible since eigendecomposition is $O(p^3)$. Interestingly, the above bias $||\Phi(x) - P_{\hat{S}}\Phi(x)||$ depends on the eigenvalue decay rate [176], that in turn depends on the data distribution and the kernel function. Hence this error in estimation is hard to predict beforehand.

Additionally, another cause of estimation error, specifically for KLSH is due to the fact that KLSH relies on the CLT in Hilbert space to generate the random Gaussians in the kernel induced feature space. Unlike the single dimensional CLT, Hilbert space's CLT's convergence rate could be much worse [120], implying that the sample size requirement may be quite high. However, the number of available samples is limited by p (number of sampled columns). Typically p is set very small for performance consideration (in fact we found that p=128 performs extremely well for dataset size up to one million).

We next propose a transformation over the Nyström embedding on which the SRP technique can be effectively used to create LSH that approximates the input kernel $\kappa(.,.)$ (K) well. Our methods apply to centered KPCA case as well.

3.4 Augmented Nyström LSH Method (ANyLSH)

In this section, we propose a data embedding that along with the SRP technique forms an LSH family for the RKHS. Given n data points and p columns of the kernel matrix, we first propose a $p+n$ dimensional embedding for which the bias is 0 (LSH estimator is an unbiased one for the Nyström approximated kernel). Since $p+n$ dimensional embedding is infeasible in practice due to large n, we propose a $p+c$ dimensional embedding, where c is

a constant much smaller than n. In this case, the estimator is biased, but that bias can be bounded by setting c and this bound hence is independent of the eigenvalue decay rate of the covariance operator. We provide theoretical analysis regarding the preservation of the LSH property and we also give the runtime and memory cost analysis.

3.4.1 Locality Sensitive Hash Family

We identify that the major problem with using Nyström embedding for LSH is the underestimation bias of the norms ($\widehat{K_{i,i}}$) of these embedding. Hence, though the estimates of the numerator of equation 3.6 are very good, the denominator causes estimation bias. We propose a new embedding of the data points such that the numerator will remain the same, but the norms of the embedding will become 1.

Definition 3.4.1. We define the augmented Nyström embedding as the feature map $Z_{n\times(p+n)}$ such that $Z_{n\times(p+n)} = [Y_{n\times p}\ V_{n\times n}]$, where $V_{n\times n}$ is an $n \times n$ diagonal matrix with the diagonal elements as $\left\{\sqrt{1-\sum_{j=1}^{p} Y_{i,j}^2}\right\}_{i=1}^{n}$.

Lemma 3.4.1. *For $Z_{n\times(p+n)}$, if $\widehat{K_{(Z)n\times n}}$ is the inner product matrix, then for (i) $i = j$, $\widehat{K_{(Z)i,j}} = 1$ and (ii) for $i \neq j$, $\widehat{K_{(Z)i,j}} = \widehat{K_{i,j}}$*

Proof. Case (i):

$$\begin{aligned}
\widehat{K_{(Z)i,j}} &= Z_i Z_j^T \\
&= \sum_{k=1}^{p} Y_{i,k}^2 + \sum_{l=1}^{n} V_{i,l}^2 \\
&= \sum_{k=1}^{p} Y_{i,k}^2 + \left(\sqrt{1-\sum_{j=1}^{p} Y_{i,j}^2}\right)^2 \\
&= 1
\end{aligned}$$

Case (ii):

$$\begin{aligned}
\widehat{K_{(Z)i,j}} &= Z_i Z_j^T \\
&= \sum_{k=1}^{p} Y_{i,k} Y_{j,k} + \sum_{l=1}^{n} V_{i,l} V_{j,l} \\
&= \sum_{k=1}^{p} Y_{i,k} Y_{j,k} + 0 \ (V \textit{ is a diagonal matrix}) \\
&= Y_i Y_j^T \\
&= \widehat{K_{i,j}}
\end{aligned}$$

□

Hence Z_i gives us a $p+n$ dimensional embedding of the data point X_i where Z_i approximates $\Phi(X_i)$. The inner product between two data points using this embedding gives the cosine similarity as the embedding are unit norm and the inner products are exactly same as that of Nyström approximation. Hence we can use SRP hash family on $Z_{n\times(p+n)}$ to compute the LSH embedding related to cosine similarity. Essentially we have:

$$P(h(Z_i) = h(Z_j)) = 1 - \frac{cos^{-1}(\widehat{K_{i,j}})}{\pi} \tag{3.7}$$

Hence we are able to achieve the LSH property of the goal equation 3.3.

3.4.2 Quality Implications

The quality of an LSH estimator depends on (i) similarity and (ii) number of hash functions. It is independent of the original data dimensionality. From equation 3.1, it is easy to see that each hash match is a i.i.d. Bernoulli trial with success probability $sim(x,y)$ (s). For k such hashes, the number of matches follow a binomial distribution. Hence the LSH estimator $\hat{s}$ of equation 3.2 is an MLE for the binomial proportion parameter. The variance of this estimator is known to be $\frac{s(1-s)}{k}$. Therefore, even with the increased dimensionality of $p+n$, the estimator variance remains the same.

3.4.3 Performance Implications

The dot product required for a single signed random projection for Z_i can be computed as follows:

$$\begin{aligned} Z_i r_j^T &= \sum_{l=1}^{p+n} Z_{i,l} R_{j,l} \\ &= \sum_{l=1}^{p} Y_{i,l} R_{j,l} + \sum_{k=1} n V_{i,k} R_{j,p+k} \\ &= \sum_{l=1}^{p} Y_{i,l} R_{j,l} + V_{i,i} R_{j,p+i} \end{aligned}$$

Hence there are (p+1) sum operations ($O(p)$). Though $Z_i \in R^{p+n}$, the dot product for SRP ($Z_i r_j^T$) can be computed in $O(p)$ (which is the case for vanilla LSH). Since $V_{n\times n}$ is a diagonal matrix, the embedding storage requirement is increased only by n (still $O(np)$). However, the number of $N(0,1)$ Gaussian samples required is $O(k(p+n))$, whereas in the case of vanilla LSH, it was only $O(kp)$ (k is the number of hash functions). In the next section, we develop an algorithm with a probabilistic guaranty that can substantially reduce the number of hashes required for the augmented Nyström embedding.

3.4.4 Two Layered Hashing Scheme

Next we define a $p+c$ dimensional embedding of a point X_i to approximate $\Phi(X_i)$. The first p dimensions contain projections onto p eigenvectors (same as first p dimensions of Z_i). In the second step, the norm residual (to make the norm of this embedding 1.0) will be randomly projected to 1 of c remaining dimensions, other remaining dimensions will be set zero.

Definition 3.4.2. Remapped augmented Nyström embedding is an embedding $Z'_{n\times(p+c)}$ ($\forall i, Z'_i \in R^{p+c}$) obtained from $Z_{n\times(p+n)}$ ($\forall i, Z_i \in R^{p+n}$) such that, (i) $\forall j \leq p$, $Z'_{i,j} = Z_{i,j}$ and (ii) $Z'_{i,p+a_i} = Z_{i,p+i}$, where $a_i \sim unif\{1,c\}$.

Definition 3.4.3. $C(i,j)$ is a random event of collision that is said to occur when for two vectors $Z'_i,\ Z'_j \in Z,\ a_i = a_j$.

Since this embedding is in R^{p+c} rather than R^{p+n}, the number of $N(0,1)$ samples required will be $O(k(p+c))$, rather than $O(k(p+n))$. Next we show that using SRP on $Z'_{n\times(p+c)}$ yields LSH embedding, where the estimator converges to $\widehat{K_{n\times n}}$ with $c \to n$.

Lemma 3.4.2. *For $Z'_{n\times(p+c)}$, the collision probability will be*

$$P(h(Z'_i) = h(Z'_j)) =$$
$$\frac{1}{c}\Big[1 - \frac{cos^{-1}(\widehat{K_{i,j}} + \sqrt{1-\sum_{l=1}^{p} Y_{i,l}^2}\sqrt{1-\sum_{l=1}^{p} Y_{j,l}^2})}{\pi}\Big] + \frac{c-1}{c}\Big[1 - \frac{cos^{-1}(\widehat{K_{i,j}})}{\pi}\Big]$$

Proof. For the remap we used, collision probability is given by,

$$P(C(i,j)) = \frac{1}{c} \tag{3.8}$$

If there is a collision, then the norm correcting components will increase the dot product value.

$$P(h(Z'_i) = h(Z'_j)|C(i,j)) =$$
$$1 - \frac{cos^{-1}(\widehat{K_{i,j}} + \sqrt{1-\sum_{l=1}^{p} Y_{i,l}^2}\sqrt{1-\sum_{l=1}^{p} Y_{j,l}^2})}{\pi} \tag{3.9}$$

If there is no collision, LSH will be able to approximate the Nyström method.

$$P(h(Z'_i) = h(Z'_j)|\neg\, C(i,j)) = 1 - \frac{cos^{-1}(\widehat{K_{i,j}})}{\pi} \tag{3.10}$$

We can compute the marginal distribution as follows:

$$P(h(Z'_i) = h(Z'_j)) = P(h(Z'_i) = h(Z'_j)|C(i,j))P(C(i,j))$$
$$+ P(h(Z'_i) = h(Z'_j)|\neg\, C(i,j))P(\neg C(i,j))$$

Applying equations 3.8,3.9 and 3.10 above, we get the result. □

There are two aspects to note about the aforementioned lemma:

1. According to Nyström approximation [159], as we increase p (higher rank approx.), the quantity $\sqrt{1-\sum_{l=1}^{p} Y_{i,l}^2}$ tends to 0 and the lemma leads to the desired goal of equation 3.3, but at a computational cost of $O(p^3)$ for the eigendecomposition operation. Of course increasing p improves the overall quality of Nyström approximation itself, however in practice small values of p suffice.

2. Interestingly, instead of p, if we increase c, then also we converge to the goal of equation 3.3 as the first term of the lemma converges to 0. The computational cost is $O(k(p+c))$ which usually is much less than $O(p^3)$. This is the strategy we adopt and as we will show shortly, small values of c are sufficient even for large scale datasets. Hence c can be used to bound the bias (difference from the probability of equation 3.3).

3.5 Evaluation

3.5.1 Datasets and Kernels

We evaluate our methodologies on five real world image datasets varying from 3030 data points to 1 million and three popular kernels known to work well on them. Summary of the datasets can be found in Table 3.2.

Caltech101: This is a popular image categorization dataset [57]. We use 3030 images from this data. Following KLSH [81, 92] we use this dataset with the CORR kernel [168].

PASCAL VOC: This is also an image categorization dataset [56]. We use 5011 images from this data. Following [37] we use the additive χ^2 kernel for this data.

Dataset	Size	Kernel
Caltech101	3030	CORR
PASCAL VOC 2007	5011	Additive χ^2
Notre Dame image patches	468159	Gaussian RBF
INRIA holidays SIFT-1M	1000000	Additive χ^2
INRIA holidays GIST-1M	1000000	Additive χ^2

Table 3.2: dataset and kernel details

Notre Dame image patches: This dataset contains 468159 small image patches of Notre Dame [67] and the image patch descriptors used are as per [141]. We use the Gaussian RBF kernel on this data.

INRIA holidays: To test at large scale, we use 1 million SIFT as well as 1 million GIST descriptors from the INRIA holidays dataset [80]. Following KLSH [81, 92] we use the additive χ^2 kernel with this data.

3.5.2 Evaluation Methodology

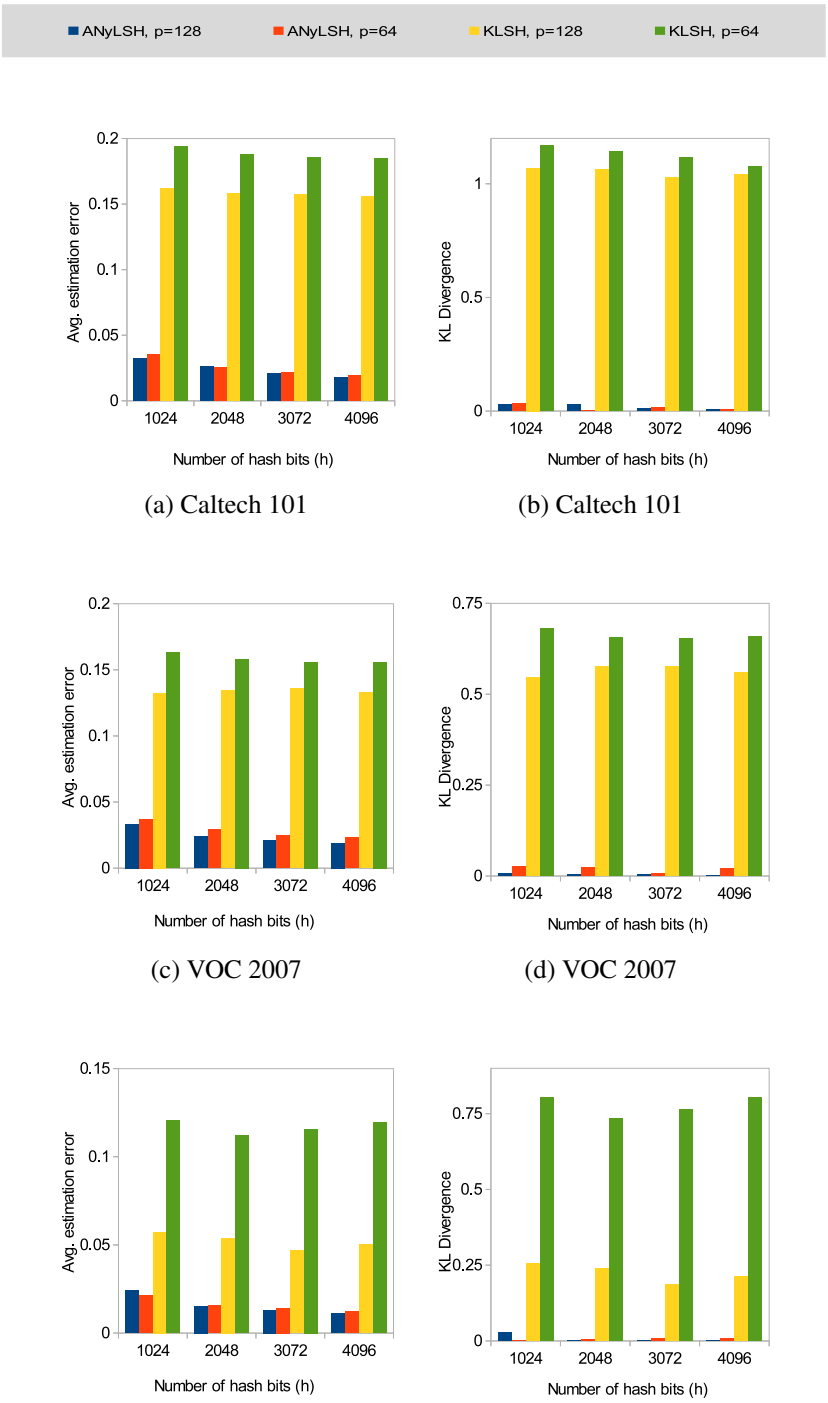
The focus of this work is accurate estimation of the input kernel similarity measure through LSH. For evaluating the quality of similarity estimation, we use two approaches - (i) we take a sample of pairs from each dataset, and compute the average estimation error directly and (ii) we use a sample of pairs from each dataset, compute the similarity of the pairs, both accurately (ground truth) and approximately (ANyLSH) and then compare the statistical distribution of the pairwise similarity of ground truth with ANyLSH. The former gives a direct measure of estimation accuracy, while the latter gives us insights on how well the pairwise similarity distribution is preserved. In terms of execution times, our algorithm performs the same as the baseline we compare against.

We use state-of-the-art KLSH as our baseline. We randomly sample 1000 pairs of data points from each dataset for our experiments. We use the values 64 and 128 for p, and vary h from 1024 to 4096 in steps of 1024. For ANyLSH, we set $c = 1000$. In our evaluation, we see that c generalizes well to varying data sizes. For KLSH, we set $q = 16, 32$ for $p = 64, 128$ respectively as per the guideline in the source code [92].

3.5.3 Results

3.5.3.1 Similarity Estimation Comparisons

Figures 3.1a,3.1c,3.1e,3.1g and 3.1i report the results on estimation error. We clearly see that our ANyLSH method outperforms KLSH in every single case by a large margin. The improvement of estimation error varies from a minimum of 2.4x (Figure 3.1e, $p = 128, h = 1024$) to a maximum of 9.7x (Figure 3.1e, $p = 64, h = 4096$), with average reduction in error of 5.9x across all datasets. With fixed p, the estimation error of our method decreases consistently across all datasets with the increase of hashes, as should be the case per equation 3.2. Interestingly, for KLSH, there are multiple cases when with the increase in hashes, the estimation error also increased. For instance, in Figure 3.1i, at $p = 64$, by increasing h from 2048 to 4096, KLSH's error increased from 0.076 to 0.078. This provides empirical evidence as well that not only the estimates are off, but in the case of KLSH, they are converging towards a biased value as described in Lemma 3.3.1. Additionally, note that our average absolute error varies between $0.011 - 0.038$ across all datasets and there is no trend that the error increases with larger datasets. This provides strong empirical evidence to the theoretical insight that at fixed c (1000 in our case), the average estimation error generalizes extremely well to different datasets of varied sizes and different kernels. Though the error is a function of the eigenvalue decay rate, it is upper bounded by ANyLSH.



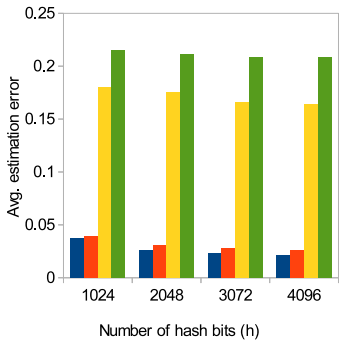
(a) Caltech 101 (b) Caltech 101

(c) VOC 2007 (d) VOC 2007

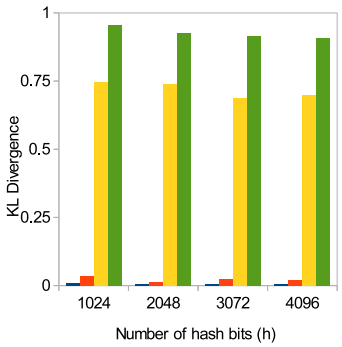
(e) PATCH (f) PATCH

Figure 3.1: Estimation error and KL Divergence are reported in the first and second columns respectively for all datasets (CONTINUED).

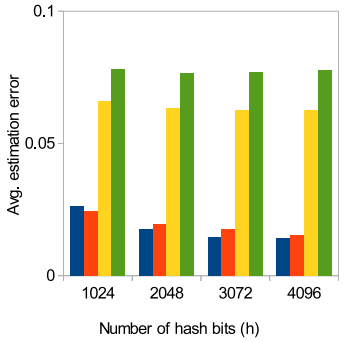
Figure 3.1: CONTINUED



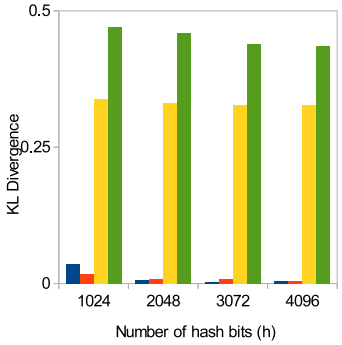
(g) SIFT-1M



(h) SIFT-1M



(i) GIST-1M



(j) GIST-1M

3.5.3.2 Similarity Distribution Comparisons

As the second part of our qualitative evaluation, in this section, we investigate how well the pairwise similarity distribution of the data is preserved. This is particularly important in applications that rely heavily on similarity distribution such as clustering. Our goal is to compare the two distributions in a non-parametric fashion as we do not have any prior knowledge of these distributions. Our first approach is to compare normalized histograms (probabilities). We choose the popular KL divergence measure to compare probability distributions represented by histograms. We discretized both our data and the ground truth by splitting the similarity range $0-1$ into fixed length bins of length 0.1. Figures 3.1b,3.1d,3.1f,3.1h and 3.1j report the KL divergence numbers. The improvement in terms of KL divergence is even better, with upto two orders of magnitude improvement over KLSH. This improvement can be partly attributed to the discretization process – since we used length 0.1 bins and our estimation errors are significantly less than 0.1, most of our errors get absorbed in the discretization process. With KLSH's error being substantially higher than 0.1, it's KL divergence becomes very high.

To account for the binning issue, we additionally run the non-parametric Kolmogorov-Smirnov two sample test that is more suitable for comparing empirical distributions of continuous data. This test is particularly challenging in our setting as the test statistic is the supremum of absolute differences across all values in the empirical CDFs. Thus, error in even a single region may result in the failure of this test. Moreover, our proposed method being an approximate one will always have some estimation error. The null hypothesis is that the two samples come from the same underlying distribution and the alternative hypothesis is that they are from different distributions. The results for $p = 128$ and $h = 4096$ are are reported in Table 3.3. All of the datasets (only exception being Caltech) did not reject

Dataset	p-value	Test statistic
Caltech101	0.006	0.076
PASCAL VOC 2007	0.716	0.031
Patch	0.565	0.035
INRIA SIFT-1M	0.603	0.034
INRIA GIST-1M	0.011	0.072

Table 3.3: Results of Kolmogorov Smirnov tests on ANyLSH method. Critical Value at 1% significance level was 0.073.

the null hypothesis, providing strong evidence that they are indeed from same underlying distribution. Note that, even the Caltech result was very close to the threshold. For KLSH, in every single dataset, the null hypothesis was rejected and the $p-values$ were extremely far away from the threshold. This conclusively proves that applying KLSH to a dataset significantly changes the pairwise similarity distribution.

3.5.3.3 Case Study: Image Search

In this section, we do a qualitative study to evaluate the efficacy of our hashing scheme. Since kernel similarity measures are known to be extremely effective in the image search domain, we choose that as an application. We use the MNIST image dataset [95]. This dataset contains 60,000 images of handwritten digits. The goal of the image search application is, *given a database of images, and a query image, find the images in the database those look similar to the query image.* We randomly choose 4 images from the dataset as query images and the rest is used as the image database. We choose the Gaussian RBF kernel as the choice of similarity measure as it is known to achieve high classification accuracy when used in kernel SVM. However, our task is unsupervised, unlike svm, we do not use any label information.

In section 3.4.4 we showed that our novel data embedding 3.4.2 can result in a correct similarity estimator if there is no collision (equation 3.10) and an incorrect estimator if there is a collision (equation 3.10) in our remapped augmented embedding. To alleviate the errors due to collision, we repeat the remapping procedure several times (5 in this experiment) and run $top-k$ nearest neighbors search each time. Finally, we generate the mean ranking and take the $top-k$ from it. Since our remapping is done uniformly at random, the effect of the bias in estimation due to collision will be alleviated across the 5 runs. We use $k=3$ for our case study. Note that this extra computation is required for the augmented part of the embedding only. The computations (hash function generation and dot product computations) related to the Nyström part of the embedding still needs to be done only once. Additionally, these 5 runs can be executed in parallel as there is no dependency. Figure 3.2 displays the results of the image search application using our approximate similarity estimator. For each row in the figure, the first column shows the query image and columns two through four show the top three nearest neighbors retrieved using our approximate similarity estimator. In all cases, we see that correct digits are retrieved.

(a) Query (b) NN-1 (c) NN-2 (d) NN-3

(e) Query (f) NN-1 (g) NN-2 (h) NN-3

(i) Query (j) NN-1 (k) NN-2 (l) NN-3

(m) Query (n) NN-1 (o) NN-2 (p) NN-3

Figure 3.2: Image search on the MNIST dataset.

3.6 Future Works

There has been a wide range of works that build on the KLSH foundations - improve quality through supervised learning [102, 115]; develop LSH for non-metric measures [116]; We believe that these methods can be used in conjunction with our hashing scheme as well to improve performance, and in future, we propose to investigate them. Additionally, we plan to explore the case of non-normalized kernel measures. Though LSH is known not to exist in the general case for maximum inner product search, but augmented data embedding along with modified LSH functions [118, 139] are known to work well for maximum inner product search. We believe these ideas can be leveraged by our data embedding framework to handle kernel similarities for the general case.

3.7 Conclusion

In this chapter we proposed a locality sensitive hash family for arbitrary normalized kernel similarity measures. We analytically showed that the existing methods of LSH for kernel similarity measures based on KPCA/Nyström projections suffer from an estimation bias, specific to the LSH estimation technique. In other words, these LSH estimates differ from the KPCA/Nyström estimates of the kernel. This bias depends on the eigenvalue decay rate of the covariance operator and as such unknown apriori. Our method, ANyLSH, can directly estimate the KPCA/Nyström approximated input kernel efficiently and accurately in a principled manner. Key to our method are novel data embedding strategies. We showed that, given p columns of the input kernel matrix, the bias can be completely removed by using a $p+n$-dimensional embedding. Since n can be rather large and also not fixed, we additionally propose a $p+c$-dimensional embedding where c is fixed and much smaller than n. In our analysis we showed that in this case the worst case bias can be controlled by the

user by setting c. Consequently, we overcame the short coming that resulted from the bias term being unknown to the user apriori. Our methods, when compared to the state-of-the-art KLSH improves the kernel similarity estimation error by upto 9.7x. Further evaluations based on the KL divergence and Kolmogorov-Smirnov tests provide strong evidence that pairwise similarity distribution is well preserved by ANyLSH.

Chapter 4: Distributed Anomaly Detection for Large-Scale Sensor Data

In chapters 2 and 3, we described similarity preserving data sketching (data size reduction) techniques and their usage to scale analytics to large datasets. In this chapter, we develop an approximate graph construction technique, that can improve the computational complexity of the Loopy Belief Propagation algorithm. This input data approximation technique to reduce the task complexity is also application specific in nature. Here we deal with the problem of distributed anomaly detection in sensor networks.

4.1 Introduction

Sensors have become commoditized, and the resulting decreases in cost have led to their deployment in large numbers. The data collected from these sensors is processed and analyzed for monitoring and managing infrastructure and resources. Anomaly detection techniques applied to the collected data allow failures and degradations to be quickly identified, enabling energy savings and more efficient consumption of resources. Some examples, where sensors in large numbers are being increasingly used, include: 1) Smart campuses and buildings, where sensors measure air temperature, humidity, building occupancy, lighting, etc., to manage heating, ventilation, air conditioning (HVAC) and lighting systems, and to optimize use of resources such as electricity, gas, and water; 2) the transportation industry,

where, for instance, vehicle fleet management is aided by continuous input from sensors attached to the vehicles. This helps in vehicle maintenance, route optimization, driver safety, etc.; 3) utility infrastructure, where sensors monitor power and water infrastructure, for efficient resource management; 4) weather modeling and forecasting, where data from a large number of distributed sensors measuring quantities like temperature, humidity, precipitation, wind speed, etc., is used for short and long term weather prediction, and aids with critical tasks such as disaster management planning, route optimization for air, ground and sea transportation.

In all the above applications, outlier or anomaly detection in the sensor data is an important task as it helps to identify abnormal conditions that may result from a failure (or impending failure), misconfiguration or malicious activity. The most commonly used outlier detection methods are threshold-based. However, a large category of anomalous behavior does not manifest as a violation of a threshold. These are addressed through model-based outlier detection techniques that build statistical models of sensors [170], typically using historical data. Most anomaly detection models have the following shortcomings — 1) the inputs to the model are local, that is, even when a large number of sensors are deployed, the global structure and dependencies between these sensors are not leveraged; 2) the models are not very robust, that is, if the input to the models is erroneous or corrupted, they typically fail to function correctly; and finally, 3) they do not perform well when there is a large amount of missing data.

In this chapter, we develop a technique, based on probabilistic graphical models, which utilizes the global dependency structure among the sensors to detect outliers. The resulting model is much more robust in the presence of erroneous data and missing sensor values than local models. Our method consists of creating a pairwise Markov Random Field (MRF)

model from the sensor network, where the graph structure is learned from historical data. While most applications of MRF [38, 83, 148] use heuristics and domain knowledge to determine the edge potentials, we learn edge potentials from historical data. We design the MRF topology such that it is very amenable to running the belief propagation inference algorithm on it. We develop a bipartite graphical model that can reduce the number of active trails in the MRF, making belief propagation converge faster and to more accurate results.

While our method applies to any of the application areas where sensors exhibit spatial dependency, including sensor deployments in buildings [16], we demonstrate its effectiveness on geographically distributed temperature sensors, mainly due to data availability. We apply our method on data collected from 5,288 weather stations in California, and compare against several baseline methods, based on local data, and others based on the immediate neighborhood of a sensor. Results indicate that our method outperforms the baseline methods. Both in the case of anomaly detection and missing sensor value prediction, our method improves the accuracy as much as 16% compared to the best baseline. Furthermore, we synthetically scaled the graphical model up to 500,000 nodes and nearly 12 million edges and observed that our technique scales almost linearly with the number of cores. Lastly, we generated an even larger graphical model with 5 million nodes and 120 million edges and ran our algorithm on a single configuration to show our method can handle a very large scale sensor deployment.

Specifically, the contributions of this chapter are as follows:

- We present a robust method for anomaly detection, using graphical models and belief propagation-based sensor state inference.

- We show how sensor data can be modeled as an MRF, where factors are naturally learned from data (rather than based on the experience of domain experts), and the graphical model topology facilitates inference.

- To evaluate performance at scale, we build a synthetic MRF generator that can generate arbitrarily large graphical models while preserving statistical properties of an MRF based on real sensor data.

- Results show that our method can predict missing values with 90% accuracy even when 75% of the data is missing; similarly, it can correctly identify 90% of outliers even when 50% of the sensors are anomalous.

The remainder of the chapter is organized as follows. Section 4.2 introduces background information and related work. Section 4.3 describes our methodology, including the inference algorithm, and the construction of the graphical model from sensor data. Section 4.4 explains our sensor data collection and synthetic data generation process. Section 4.5 provides our experimental results. Lastly, Section 4.6 summarizes our work.

4.2 Background and Related Works

4.2.1 Outlier detection

According to Hawkins, “An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism." [72], Typically, it is assumed that data is generated by some underlying statistical distribution and the data points that deviate from that distribution are called outliers. Outlier or anomaly detection techniques are widely used in applications such as fraud detection

in financial transactions, email spam detection and medical symptoms outlier detection. A survey of anomaly detection techniques is presented in [35].

In this work, we focus on identifying faults in sensor observations in a large-scale sensor network. [170] provides a thorough overview of outlier detection methods in sensor network data. According to this survey, the types of outlier detection methods used on sensor data include *statistical-based, nearest neighbor-based, cluster-based, classification-based and spectral decomposition-based techniques*. The outlier detection model we propose falls under the broad category of statistical-based techniques, where the sensor data generation is assumed to follow some statistical distribution and any observation that does not fit in that distribution is an outlier. The challenge is to learn the distribution of the sensor data. Local techniques require the learning of data distribution of sensors in isolation. They are only able to capture the temporal dependency in observations, while global techniques try to learn the joint distribution of the sensor data and hence are able to capture the spatial dependency as well. More precisely, our outlier detection model is based on the underlying principles of *statistical relation learning* [64], as we describe in the next section. Outlier detection models may be both supervised or unsupervised. For instance *knn* based outlier detection techniques [126] are the most popular unsupervised methods, while classification based outlier detection methods include SVMs [82], Random Forests [169] and others. Outlier detection models can be further classified into parametric and non-parametric methods. Popular parametric models include learning Gaussian-based models and doing hypothesis tests on new data [91], while non-parametric techniques include histogram learning and kernel density estimators [134]. The drawback of parametric models is that the model assumptions (such as Gaussian distributions) may not be realistic. Also, non-parametric models are easier to generalize. *Our model falls into the category of unsupervised and*

non-parametric method. We do have a training phase, however, we do not require class labels, i.e., we do not need to know whether the sensor observations are outliers or not. Our method uses a histogram based approach.

4.2.2 Statistical Relational Learning

Statistical relational learning [64] (SRL) involves prediction and inferences in the case where the data samples obtained are not, independently and identically distributed. In most real world applications, there is a complex structural relationship among the individual variables of the system and there is uncertainty in the relationship as well. SRL formalizes the structure of the relationship and accounts for the uncertainty through probabilistic graphical models. The end application such as outlier detection, link prediction, topic modeling etc. usually requires inference on the graphical model. Our problem of detecting anomalies in sensor observations fits nicely in the SRL model. Since properties such as temperatures of close by regions are related, there are strong spatial dependencies among the sensor observations. There will be temporal dependencies for observations on a single sensor as well, however, this can be studied in isolation using a local model. In this chapter, we model the global spatial dependency in the sensor network using a probabilistic graphical model and the outlier detection involves doing inference on the model.

One of the main challenges in SRL is designing the graphical model. The graph needs to capture the dependencies accurately, while not being excessively complex to make inference hard. This trade-off between accurately and cost of inference needs to be carefully made based on the requirements of the application. The two most popular graphical models are Bayesian networks and Markov networks (Markov random field). We choose a Markov network as it is undirected and there is no directionality of dependence

in the sensor observations. Bayesian networks are appropriate in relations resulting from causal dependency. The most common methods of doing inference on Markov networks are (i) MCMC based methods (e.g., Gibbs sampling) and (ii) variational methods (e.g., loopy belief propagation) [135, 164]. MCMC based techniques such as Gibbs sampling are known to be extremely slow with high mixing time, therefore requiring an extremely high number of samples. We choose the variational method called loopy belief propagation because of its simplicity and it is in many cases much faster than MCMC based methods (especially for slow mixing chains) [106]. Additionally, with careful design of the graph, the performance of the loopy belief propagation algorithm can be improved significantly. Furthermore, compared to MCMC, loopy belief propagation performs well on large scale, sparse graphical models, such as those constructed from sensors.

Another related work [165] for spatiotemporal event detection uses conditional random fields (DCRF) and belief propagation for the inference. However, DCRFs assume a specific functional form of the factors (potentials) that can be factorized over a fixed number of features [147]. We take a nonparametric approach in this work. Additionally, we learn the spatial dependency graph structure whereas the aforementioned work fixes the graph and can only learn the parameters. A graphical model based approach has also been used to forecast wind speeds [84], but again a strong assumption of Gaussian processes has been made. In our target application of sensor data analytics we will discuss methods of designing the MRF such that belief propagation converges faster empirically. A survey of anomaly detection techniques based on graphical representation of data, including graphical models is presented in [5].

4.3 Methodology

4.3.1 Markov Random Field

A probabilistic graphical model (PGM) [89] combines probabilistic dependency and reasoning with graph theory. Often to simplify a modeling problem, random variables are assumed to be independent, resulting in loss of information. PGMs provide a means to express and reason about these dependencies without making the problem overly complex (e.g., assuming every variable depends on all others). The graph structure provides an elegant representation of the conditional independence relationships between variables. We model the data generative process of a large group of sensors as a Markov random field (MRF), an undirected PGM where each node in the graph represents a random variable and edges represent the dependencies between the random variables. A factor or potential function specifies the relationship between variables in each clique. The joint distribution of all of the random variables in a model can be factorized into the product of the clique factors through the Hammersley-Clifford theorem [130].

$$P(x_1, x_2, ..., x_n) = \frac{1}{Z} \prod_{c \in \mathscr{C}} \phi_c(X_c)$$

where x_i are the random variables; $\mathscr{C}$ is the set of all maximal cliques in the graph; ϕ_c is the factor for clique X_c; and Z is the normalizing constant, also called the partition function. Markov Random fields have three important characteristics; (i) Pairwise Markov Property: Two random variables (represented by nodes in the graph) are conditionally independent given all other random variables $x_i \perp\!\!\!\perp x_j | G \setminus \{x_i, x_j\}$; (ii) Local Markov Property: A random variable (represented by a node in the graph) is conditionally independent of all other random variables given its direct neighbors in the graph $x_i \perp\!\!\!\perp G \setminus \{x_i \cup nbr(x_i)\} | nbr(x_i)$, where the

set nbr(x_i) contains the neighbors of x_i. (iii) Global Markov Property: Two groups random variables (represented by nodes in the graph) are conditionally independent given a set of nodes that disconnects the groups of nodes $X_i \perp\!\!\!\perp X_j | X_k$, where X_i, X_j and X_k are mutually exclusive groups of nodes and removal of X_k disconnects X_i and X_j.

The Hammersley-Clifford theorem allows the joint probability distribution of the graph to be factorized into a product of the joint distributions of its cliques. This is why we chose a pairwise Markov Random Field (clique size of 2) as it helps bring the down the model complexity significantly making inference scalable. Secondly, from the three MRF properties we observe that between two the nodes, if all the paths (called active trails) can be broken, then by virtue of conditional independence, the model will be further simplified making it scalable to huge graphs. We achieve this by designing the MRF graphical model such that many of the active trails are broken making the inference procedure fast.

4.3.2 Loopy Belief Propagation

Belief propagation [122] is a commonly used, message-passing based algorithm for computing marginals of random variables in a graphical model, such as a Markov Random Field. It provides exact inference for trees and approximate inference for graphs with cycles (in which case it is referred to as loopy belief propagation). Even though loopy belief propagation is an approximate algorithm with no convergence guarantees, it works well in practice for many applications [117] such as coding theory [39], image denoising [60], malware detection [38, 148]. The loopy belief propagation algorithm to compute marginals is shown in Algorithm 1.

The algorithm works in two steps - (i) based on the messages from the neighbors, a node updates its own belief (line 5) and (ii) based on its updated belief, a node sends out

1 Initialize all messages to 1;
2 **while** *all messages not converged* **do**
3 | **for** $i \in$*{active vertices}* **do**
4 | | read messages $m_{ji}(v_i),\ \forall j \in NB(i)$;
5 | | update own belief, $b(v_i) = g(v_i)\prod_{j \in NB(i)} m_{ji}(v_i)$;
6 | | send updated messages $m_{ij}(v_j) = \sum_{v_i} \frac{b(v_i)}{m_{ji}(v_i)} \phi_{ij}(v_i, v_j),\ \forall j \in NB(i)$;
| | **end**
end

Algorithm 1: Loopy Belief Propagation

messages to its neighbors (line 6). In the update step (i), a node i's belief $b(v_i)$ is a product of its prior belief $g(v_i)$ and messages received from its direct neighbors. $\{v_i\}$ is a set that represents the discrete domain of the probability distribution of random variable i. In the send step (ii), a node i generates a new message for node j for value v_j by marginalizing over the other values $v_i (\neq v_j)$. Note that in this step when a node i is creating a message for node j, it ignores the message it received from node j itself. This is achieved by dividing the factor by $m_{ji}(v_i)$. These aforementioned steps are repeated until convergence (line 2 and 3). Convergence is achieved when there are no active vertices left. An active vertex is one whose belief has not converged yet.

The computational cost of a single iteration of Loopy Belief propagation is $O(n+e)$, where n is the number of nodes and e is the number of edges. Note that in reality, the computational cost does not depend on all nodes, rather for a specific iteration, it is only dependent on the active vertices and edges between them. Usually, the number of active vertices become much smaller as the iterations progress. Since LBP does not have a convergence guarantee, it is not possible to establish overall computational complexity. However, we design the dependency graph restricting active trails and empirically show

that this results in much faster convergence, allowing scale up to large graphs containing millions of nodes.

4.3.3 Graphical Model Design

While our method is generic and applies to any sensor data, in this chapter we use real data from temperature sensors to detect outliers. We approach the problem by building a generative model of the temperature sensor data. Given the observations or evidence, this model can be used to predict the temperature state of a particular sensor. We inject anomalies in sensors, and then make predictions using our model. If there is a large discrepancy between the observed value and the predicted value of a sensor, we conclude the sensor observation is an outlier. We can learn a similar model, e.g., a regression model, using features available locally at a sensor, or even features obtained from sensor neighborhood, however, that model will not be able to exploit the global structure of the graph, particularly when a large number of observations are missing.

4.3.3.1 Learning the dependency graph topology

We learn the spatial dependency graph structure from historical data. This is an offline process. For each pair of sensors, we compute the frequency of the co-occurring observations using the historical data and normalize it to a 0-1 scale. We call this the dependency score. For each pair of vertices, we get a vector of such scores. If the maximum value in the dependency score vector exceeds a certain threshold, we keep the edge between the pair of vertices. This threshold parameter is experimentally determined. For our specific temperature sensor network application, we use fixed width binning to discretize the temperature domain into sixteen intervals of five degrees F each. Since each sensor can

report one of the sixteen states, a pair of sensors can report $16X16 = 256$ jointly occurring states. In other words, the dependency score vector is of length 256.

4.3.3.2 Creating Markov Random Field

Given a graph topology, we now need to convert it to a pairwise Markov Random Field. Since the factors in a pairwise MRF essentially capture the dependency among the nodes and need not be proper probability distributions, we can directly use the dependency score vectors as the factors in the MRF. This gives us a way for factor construction for each edge in the MRF. In terms of the vertices, every sensor node in the dependency graph has two corresponding vertices in the MRF: 1) observed temperature of that sensor (observed state); and 2) true temperature at that location (hidden state). Next, we show two different MRF graph construction designs and discuss the benefits of the design choices. Figures 4.1 and 4.2 show the two design choices for the graph construction. The circular even-numbered nodes represent the random variables for hidden true states of the sensors and the square odd-numbered nodes represent the random variables for the observed states of the sensors. Every sensor is hence represented by two random variables.

MRF Topology 1: In the first design choice (Figure 4.1), if there is an edge between two sensor nodes in the dependency graph, we add an edge between the corresponding true state nodes in the MRF. The factor on that edge will be the dependency score vector between the pair of sensor nodes in the original graph. This implies the true temperatures of the sensor locations are related according to the learned dependency graph, and the observed state of every sensor depends on the true state of that location. For every sensor node in the original dependency graph, we add an edge between the true state and the observed state of the same sensor in the MRF. Thus, if there were N nodes and E edges in the original graph, the pairwise MRF will contain $2N$ nodes and $E+N$ edges.

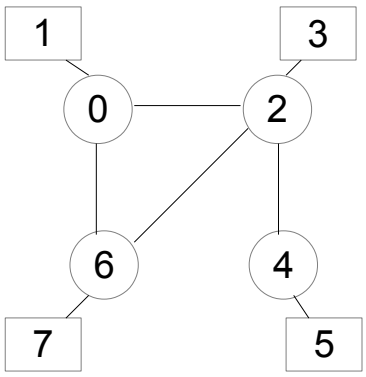


Figure 4.1: MRF Topology 1

MRF Topology 2: In the second approach (Figure 4.2), instead of the true states of each sensor location being dependent on each other, the true state of a sensor location depends on the observed states of its' neighboring sensors. In other words, we form a bipartite graph of the true states and the observed states. For every sensor node in the original dependency graph we introduce an observed state and true state for that sensor in the MRF. We connect the observed state and true state as before. For every edge i, j in the dependency graph, we add an edge between the observed state of i and the true state of j, and another edge between the observed state of j and the true state of i to form the bipartite graph.

4.3.3.3 Discussion

At first glance, design choice 1 seems to be the better choice as it captures the intuitive dependencies. Specifically, the true temperatures of the nearby sensor locations should be heavily dependent and the observed state of any sensor should be highly dependent on the

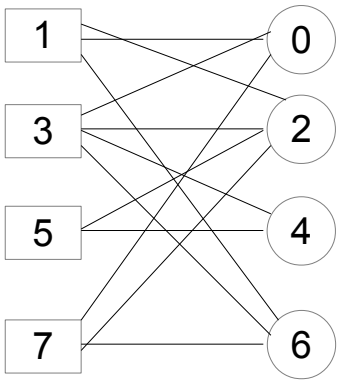


Figure 4.2: MRF Topology 2

true state of the location. This is exactly the dependency captured by the first design choice. However, this design choice has some practical problems. Since all of the random variables corresponding to the true states are hidden (not observed) and these are the ones connected in the MRF, the MRF will have a lot of loops with hidden states. In other words, there will be more and longer active trails in this MRF design. As a result, the loopy belief propagation algorithm is likely to take a long time to converge. In fact, in presence of loops, the belief propagation algorithm does not have convergence guarantees. Even if it converges, it could be to a local optima. This is why design choice 1, although intuitively very appealing is not a very practical one. Design choice 2 on the other hand largely overcomes the shortcomings of choice 1. Since we formed a bipartite graph between the hidden random variables and the observed ones, the presence of an observation on the observed random variables breaks the active trail in the loop. Thus, as the number of observations increase, the number of active trail loops in the MRF decreases. The best case scenario is when we have observations

	Topology 1	Topology 2
Accuracy (%)	88.9	97.5
Time (secs)	663.5	27.6

Table 4.1: 5% missing value prediction

for all the designated observed state variables (when all sensors are functioning). In this situation there are no active trail loops, and as a result belief propagation converges in two steps. As long as the observed values are correct, the converging state is the global optima. Note that, in practice, a significant number of observations may be missing. Even as missing data increases, design 2 still will have less active trails as compared to design 1 and it will converge faster to more accurate results. We empirically verify this by running belief propagation on a single instance of a test graph generated from real sensor data, described in the next section. We evaluate both design choices under two configurations – 5% missing value and 50% missing value. The results are listed in Table 4.1 and Table 4.2, respectively.

Note that when there is no missing sensor data, exact inference is possible in topology 2 by considering the joint probability distribution of a node and its neighbors. The joint distribution can be factored into pairwise factors and then marginalized to get the probability distribution of the node. When missing data is low, this approximation can be used at the cost of some accuracy.

	Topology 1	Topology 2
Accuracy (%)	85.1	97.6
Time (secs)	60.1	45.5

Table 4.2: 50% missing value prediction

4.4 Sensor Data

4.4.1 Data Collection

We chose a temperature sensor network as an application in this work as it was possible to collect a large, real data set. We identified about 5,288 California weather stations from weatherunderground.com and collected raw data from the stations for a month. We used a python script to collect raw data on a daily basis for each of the stations for the entire month. Once the raw data was downloaded, we used a second script to extract the time of day, temperature, pressure, humidity and wind speed on an hourly basis for each of these stations. We use 20 days data for training our model. We used a third script to learn the pairwise dependencies between each pair of sensors and create the dependency graph. Since we check all pairs of sensors for dependency, we parallelized the script to reduce the time required by the offline training procedure. Lastly, we used a fourth script to convert the dependency graph to the pairwise MRF (both design choices).

4.4.2 Synthetic graphical model generator

The MRF graphical model of the real California sensor network is approximately 10,576 vertices and 234,968 edges. This is still quite small to understand whether our model will scale to really large scale sensor networks that we expect will be commonplace. Therefore, we develop a synthetic graphical model generator which can generate arbitrarily large

graphical models while preserving the properties of the original (smaller) graphical model. The main requirement of our synthetic graphical model generator is that it should generate a Markov Random Field with similar joint probability distributions as the original smaller seed model. The second requirement is to create a set of observations for the MRF which can serve as the ground truth.

There is a lot of prior work on graph generation models that can preserve the topology (degree distribution), as well as co-related vertex attributes. For example, this could be done using the accept-reject sampling technique [123]. Unfortunately, these generic techniques are not suitable for generating MRFs. Specifically, MRFs are non-attributed graphs, where each vertex represents a random variable and each edge represents a factor potential table. A sensor node in a sensor network graph may observe multiple attributes, but when represented as an MRF, each attribute will become a vertex in the MRF and all dependencies will be captured by adding edge factor potential tables. We can still use standard techniques [111] to create a larger graph while preserving the degree distribution of the original MRF. To generate the edge factors and also to generate multiple instances of the graph with different observations that will follow the underlying joint probability distribution, we developed the following technique:

1. From the original dependence graph, we extract the degree distribution.

2. From the original dependence graph, we extract the dependence score vector distribution.

3. We create a larger graph where the nodes follow the same degree distribution and the edges follow the same dependence score vector distribution. The underlying assumption here is that the number of neighbors of a node is independent of the

strength of the dependency with each neighbor, which is a reasonable assumption to make.

4. We then randomly select a small percentage (less than 10) of the nodes and randomly assign temperature values to them. Again the idea is, since the percentage of selected nodes is very small, the temperatures at those locations should be fairly independent of each other. Hence randomly assigning temperatures will not conflict with the dependency structure of the graph.

5. We create an MRF (choice 2) from the synthetically generated larger graph and run belief propagation to predict the state of the entire graph. This serves as the ground truth observations. We empirically show on the real graph that with 90% missing data, the rest of the graph can be reconstructed with more than 85% accuracy. We believe the same should hold for the synthetically generated larger graph as it has the same properties of the real graph.

4.5 Empirical evaluation

We used an HP ProLiant DL980 server with 80 CPU cores to conduct our experiments. Each CPU core has a speed of 1.9 GHz. The server has 2 TB of memory. Our belief propagation algorithm is developed as a C++ application on top of an open source graph analytics framework called GraphLab/Powergraph [70, 103]. Note that although we use a large memory machine, we could have equally well run GraphLab on a cluster of commodity machines. For baselines and other functionalities such as data collection and graph generation, we use python scripts.

4.5.1 Experimental methodology

4.5.1.1 Quality evaluation

We run our belief propagation-based outlier detection algorithm on the real temperature sensor dataset. The MRF we generated had 10,576 vertices and 234,968 edges. We collected hourly observations for a month. We used the first 480 hourly observations for training the MRF (learning the graph structure and factors). We used a threshold of 0.5 for keeping an edge in the graph. In other words, after we learn the factor table for a pair of sensors, if the maximum value in the table is greater than 0.5, we introduce an edge between the pair of sensors. We also tried a lower threshold (0.4), which produces similar results but significantly increases the number of edges (over 1 million). After learning the MRF, we use the next 50 hours' observations to instantiate 50 such graphs and test our algorithm on each of the 50 instantiations.

Based on inference, our method predicts a sensor value. We evaluate our method's accuracy in two scenarios – i) when data is missing; ii) when sensor values are erroneous due to anomalies, data corruption, or malicious activity. Since a sensor value is considered an anomaly if its predicted value significantly deviates from its observed value, we use the prediction accuracy as a proxy for the accuracy of anomaly detection. Even when the number of sensors with missing data or anomalous data is large, we are still able to accurately recover their correct values, as we show in Section 4.5.

Predicting missing sensor values: To evaluate accuracy in this case, we randomly delete a percentage of sensor observations from the MRF, and then try to predict them. We do this for all the 50 test graphs we instantiated. We measure quality by the accuracy of predicting the missing values averaged over all missing values over all 50 graphs. As mentioned earlier, we divide the temperature range into 16 discrete bins. We allow the prediction to be off by at

most a single bin to be considered correct. We vary the percentage of missing observations from 5% to 90% and report quality for each setting. Note that there is already missing data in the original dataset.

Detecting anomalies/outliers: To evaluate accuracy when there is anomalous data, we randomly inject outliers to a percentage of sensor observations and then try to detect them. The outliers we inject are random temperature values. Again we predict the true state of these sensors and see whether they match (the difference can be at most one adjacent bin) with the observed values (the injected outliers). If the values do not match, then we are able to successfully detect the observation as an outlier. We report the average outlier detection accuracy over all 50 test graphs. We vary the percentage of outliers from 5% to 90% and report quality for each setting. Note that the data we introduce outliers to already has a significant amount (about 25%) of missing data as well.

4.5.1.2 Performance evaluation

We generate a larger graphical model with 500,000 nodes and 12 million edges using our synthetic graphical model generator, which takes as input the graphical model based on real data. We seed 1% of the nodes with random observations and run belief propagation to evaluate the performance. We measure performance in terms of time taken by the inference engine. We also measure the scale out performance by increasing the number of threads in GraphLab from 1 to 32 and observing the effect on the running time. We test two versions of the belief propagation algorithm, the bulk synchronous processing version, and the dynamic asynchronous version.

4.5.1.3 Baselines for quality comparisons

We compare our work against four different baselines:

1. **Local regression:** We learn a sensor specific local model, which implies each sensor's model is agnostic of the rest of the sensor network. A sensor can only use its historical data to make predictions. We learn multivariate linear regression models for each sensor. The features of the regression are *previous two hour's temperature, current pressure, current humidity and current time of day*. The model predicts the current temperature. We inject outliers in the input features (previous two hour's temperature) and see how it affects the quality. We vary the percentage of outliers from 5% to 90% and report the impact in quality. Note that this baseline is only applicable to the outlier detection application.

2. **Neighbors-mean:** Here instead of building a model based on a sensor's data alone, we build a model using a sensor's data and its neighbors' data. Note that to decide which sensors are neighbors, we still use our MRF graph learning process as described earlier. This model simply averages the observed values of its neighbors as its predicted value. This model is used for both missing value prediction as well as outlier detection. We vary the percentage for both tasks from 5% to 90% and compare the results.

3. **Neighbors-median:** This model is essentially the same as the *Neighbors-mean*. The only difference is instead taking average, this model predicts its own value as the median of its neighbors' observed values. This model is likely to be more robust in presence of outliers.

4. **Neighbors-wmean:** Here we take the mean of the neighbors, but weigh a neighbor's value by the normalized maximum value in their joint factor table.

A more complex model could be trained on a node's local and neighbors' features. However, as missing or anomalous data increases – when our method is most advantageous – the performance of such a model is likely to severely degrade.

4.5.2 Results

Figures 4.3 and 4.4 show the accuracy of correctly predicting sensor values using the baseline methods and our proposed method (marked BP) for increasing percentage of missing values and increasing percentage of anomalous values, respectively.

In the case of predicting missing sensor observations (Figure 4.3), our belief propagation-based model performs very well. The prediction accuracy decreased only 10% (from 95% to 85%) when the percentage of missing sensor observations was increased from 5% to 90%. This shows the efficacy of learning the factors of the MRF from historical data as compared to the more traditional way where a domain expert determines the graph structure and the factors. Even when only 10% of the sensors had observations available, the whole network could be reconstructed with 85% accuracy. This is due to the ability of the technique to learn the dependencies among sensors correctly through the edge factors in MRF. This is why the dependency among two sensors that are not immediate neighbors can still be modeled through the series of edge potentials connecting them. As expected, the median-based method performs better than the mean-based method. Our method is as much as 16% better than the best baseline (Neighbors-median).

The results for the case where an increasing percentage of outliers are introduced into the data set are quite interesting. We see from Figure 4.4 that our belief propagation based model's accuracy is very high (about 90%) when the percentage of injected outliers is less than 50%. Once the number of outliers increases beyond 50%, there is a substantial

degradation in accuracy. We believe this is because belief propagation is a gossip-based, message passing algorithm. As a result, once the number of outliers is a majority, the global belief converges to the outlier values. When the number of correctly observed values are a majority, most beliefs still converge to correct values. However, once the number of outliers becomes the majority, the model starts degrading rapidly. While our method performs better than the baseline methods up to around 80% outliers, beyond that it does worse. This is expected due to the global impact of the outlier nodes in BP. In the local regression based model, almost any injected outlier results in an incorrect prediction. This is why the degradation in quality is roughly linear with the increase in outlier percentage. Another point to note is that the degradation in accuracy is more severe with outliers than with missing data. This is true for both the baselines and our method. Here again, our method is as much as 16% better than the best baseline (Neighbors-median).

The performance numbers are shown in Figures 4.5 and 4.6. We vary the number of threads from 1 to 32 and report the scale-out pattern of the belief propagation inference engine on GraphLab for two MRFs - 1) 10,576 vertices and 234,968 edges real dataset and 2) 500,000 vertices and 12 million edges synthetic dataset. While the asynchronous version performs better than the synchronous version, both versions performed quite well with increasing parallelism, as shown in Figures 4.5 and 4.6. On the largest graph (5M nodes, 120M edges) the synchronous version using 32 threads completed in 12 hours. Table 4.3 provides the memory requirements of each graph. These results indicate that the memory consumption scales linearly with the size of the graphical model.

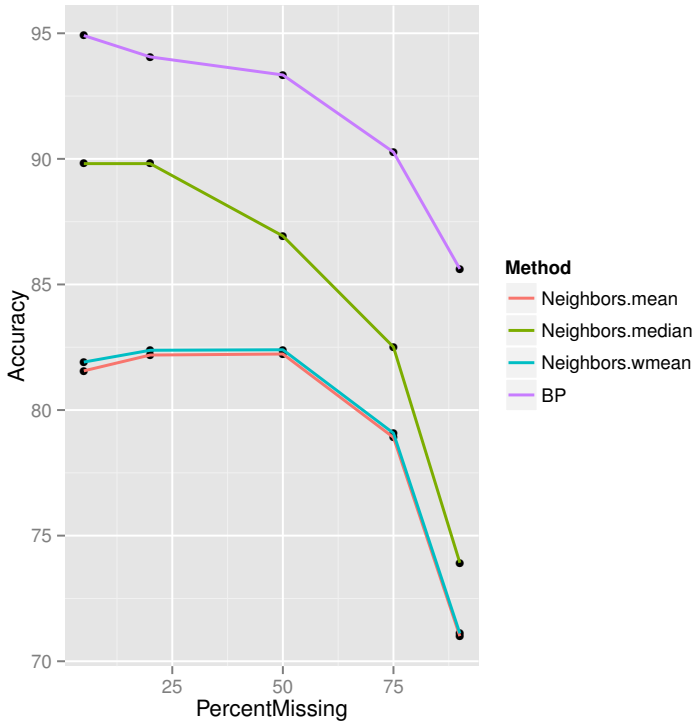


Figure 4.3: Model accuracy results – Data with missing values.

Nodes, Edges	Memory (GB)
11K, 235K	3.5
500K, 12M	160
5M, 120M	1,600

Table 4.3: Memory consumption of belief propagation

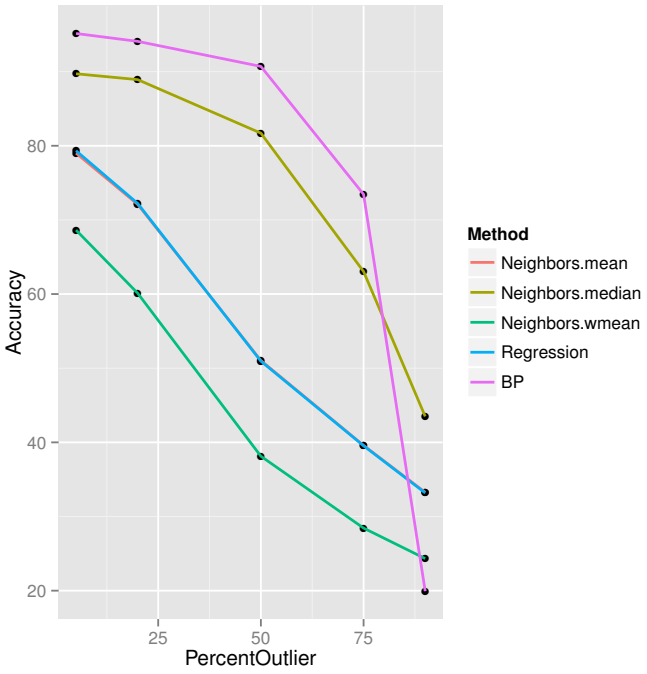


Figure 4.4: Model accuracy results – Data with anomalies/outliers.

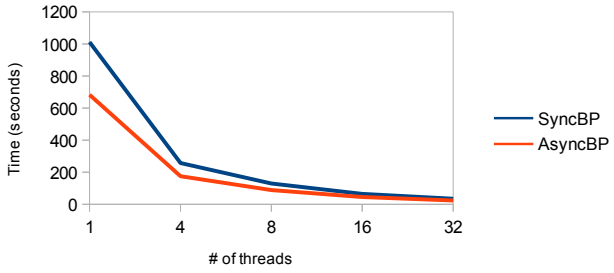


Figure 4.5: Runtimes for loopy BP on MRF with 11K nodes, 235K edges.

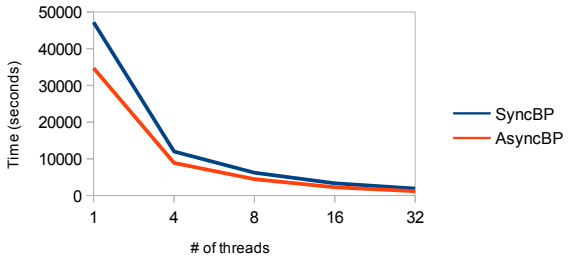


Figure 4.6: Runtimes for loopy BP on MRF with 500K nodes, 12M edges.

4.6 Conclusions

We developed techniques for anomaly detection in sensor data as well as for predicting missing sensor observations. We presented a novel way for learning the dependence structure among sensors and represented it using a bipartite Markov Random Field graphical model which is very amenable to the belief propagation inference algorithm both in terms of accuracy and performance. We tested our algorithm on real temperature sensor data collected from 5,288 California weather stations. Results showed that our method can predict missing values with 90% accuracy even when 75% of the data is missing; similarly, it can correctly identify 90% of outliers even when 50% of the sensors are anomalous. Our algorithm provides most benefit when large amounts of data is missing by exploiting transitive dependencies through use of graphical models and belief propagation. We also developed a synthetic graphical model generator which can generate a large MRF while preserving the properties of a smaller graphical model. We use it to generate 500,000 nodes and 12 million edges graphical model to show scalability of our method.

Chapter 5: A Distributed Framework for Data Analytics on Heterogeneous Systems

In this chapter, we build a general framework for distributed analytics. Our framework takes into account the processing environment's heterogeneity while partitioning and placing the data to workers. Additionally and importantly our framework is also *payload aware* which as we'll show in this chapter is key to efficient distributed execution of analytics tasks. Furthermore, the LSH techniques from chapters 2 and 3 can provide valuable insights on the characteristics of the payload and guide the partitioning strategy.

5.1 Introduction

Distributed algorithms for data analytics partition their input data across many machines. The de-facto approach typically involves a simple scheme such as random or round-robin. A partitioning scheme assigns data to machines. For example, random partitioning schemes choose host machines by sampling a uniformly random probability distribution. A data partition is the set of data assigned to a machine. The size and content of each partition can significantly affect the performance and quality of analytics. Size matters because some machines within a rack or data center perform worse than others. Slow machines should process small partitions. Typical solution in this space is a work stealing [21] approach (whichever node finishes its share of partitions, randomly selects a partition from another

slower node). However, traditional work stealing based solutions will not scale for distributed analytics workloads as these workloads are typically sensitive to the payload (content) along with the size of data. Content matters because statistical skew across partitions can slow down an analytics algorithm and degrade the quality of its results.

Consider frequent pattern mining [4]. For large datasets, the distributed algorithm simply divides the data into partitions and runs frequent pattern mining locally on individual partitions. Consequently, some of the locally frequent patterns are not globally (considering entire data) frequent and hence an additional scan of the generated candidate patterns is required to prune those. Essentially the total number of candidate patterns represents the search space – the more the number of candidate patterns, the slower the run time. These false positive candidate patterns arise due to the statistical skew across data partitions.

An approach [155] for data partitioning in the homogeneous context based on data stratification (grouping similar or related datum into strata) and then leveraging this information for data partitioning was proposed recently. However, a limitation of this approach is that it does not account for the inherent heterogeneity present in modern data centers. Most real-world data centers are increasingly heterogeneous in terms of both their computational capabilities as well as the available clean energy they use. There are many reasons for such *heterogeneity* such as equipment upgrades and power-performance tradeoffs within modern processor families. We describe the types of heterogeneity in section II.

In this work, we present a partitioning scheme for data analytics on heterogeneous clusters. Our approach scans the data before initiating the analytics workload and identifies similar content. It then builds partitions that reflect the global distribution of data and relative processing capacity of each node. Our approach overcomes several challenges. First, we significantly reduce resources needed to find similar content. We use a lightweight data

sketching approach that works in one pass using only in-memory computation. Second, we profile each machine's processing capacity by using small, samples to benchmark the target analytics workload. These samples should be representative of the partitions on which the actual algorithm will run. And then we learn the execution time objective function using progressive sampling, where the samples are representative of the partitions. We take samples of increasing size and run the actual algorithm on them to measure time. We then fit a function to predict execution time given the input data size. The dirty energy consumed by each partition depends on the execution time on each partition and the amount of renewable energy available to the physical server hosting that partition. To model the renewable energy availability we use the PVWATTS simulator [2] from NREL. Combining this with the aforementioned execution time function, we get our objective function for predicting the total dirty energy consumed by a job. Our goal is to minimize the total dirty energy consumption as well as minimize the maximum running time across all partitions. Hence we frame a multi-objective optimization problem, where the objective function is essentially a weighted average of the objective function with respect to execution time and the objective function with respect to dirty energy consumption. The weighing parameter controls the tradeoff between speed and energy consumption. The solution to the optimization problem gives the partition size distribution. This results in well load balanced execution of the job.

Specifically, the contributions of this chapter are as follows:

- We characterize the innate heterogeneity in today's cloud computing environments. We explain why special care is needed while accounting heterogeneity in the case of distributed analytics algorithms. We show that to provide time and energy aware solution, we need to model the hardware specifications as well as the underlying data distribution.

- We provide a simple yet principled approach to model the heterogeneity problem in the computing environment. We describe a method to learn the objective function with respect to the execution time by using progressive sampling technique. This method is aware of both the machine capacities and the content of data. We frame a multi-objective optimization problem for time and energy, which can be solved efficiently using linear programming technique.
- We build our partitioning framework as a middleware on top of the popular NoSQL store Redis. We conduct a thorough testing of our framework on three popular data mining workloads on five real life data sets and found out that we get up to 51% reduction in time while optimizing for time only and we get up to 31% reduction in time and 14% reduction in dirty energy consumption when we optimize them simultaneously.

5.2 Motivation

Data centers are often heterogeneous in terms of the performance of their constituent systems. This is inevitable since nodes fail periodically and are often replaced with upgraded hardware. Even when nodes have similar processing capacity, the processing rate of individual virtual machines can still vary. For example, cloud instances hosted on Amazon EC2 with the same hardware specifications exhibit 2X variation in throughput [19]. Another type of heterogeneity increasingly common to data centers is the varying dirty energy footprint of different physical servers. Some leading contemporary designs include:

1. [51] proposes to put the grid ties and renewable supplies at rack level or individual server level rather than at the data center level. This allows data centers to concentrate the green energy as much as possible to the users requesting it.

2. iSwitch [99] envisions that in future data centers, different racks will have different power sources, some might be fully powered with green or dirty energy, while some racks might be powered by both and jobs will be placed to minimize the usage from purely grid tied racks (guarantees availability).

3. Another design gaining prominence [162, 171] is the geo-distributed data centers, where jobs are scheduled to use servers from different geographical regions to maximize the use of green energy.

Additionally, a key challenge, for a large class of analytic workloads running on such data centers, is their irregular access patterns and their payload (input parameters, data skew) dependency. Simply partitioning the data while accounting for heterogeneity alone will lead to sub-optimal solutions. The reason is, the time taken to process a partition of data is dependent on the statistical distribution of data as well as the capabilities of the node (processing speed, green energy harvesting).

Here we propose a novel framework that partitions data in a payload-aware way such that the total execution time is reduced while simultaneously accounting for the dirty energy footprint of individual servers. Importantly we note that optimizing for energy (at least the way we have described it) is somewhat at odds to optimizing for performance – in other words, there is a Pareto-optimal tradeoff to be explored here [62]. To reiterate, we note that for the problem we tackle in this chapter, optimizing for energy is not equivalent to optimizing the “race to halt" (or performance) [41].

5.3 Methodology

The key elements of our methodology are: i) the specific tasks (which in the context of big data applications are irregular and data dependent in nature); ii) the heterogeneous

processing capacity within the data-center-of-interest; iii) the heterogeneous energy footprint within the data-center-of-interest; and iv) the inherent data distribution of the payload. Specifically, we seek to estimate task complexity, heterogeneous processing capability, clean energy availability within the data center, and payload characterization, respectively. We propose to estimate these quantities as follows. First, in an effort to simplify the model we couple the first two elements and seek to estimate the task complexity on a specific system (set of individual machines) by operating on a small sample problem. We note that this estimate will vary across different data analytic tasks and across different datasets (a desirable property), but is a one-time cost (small) and will be amortized over multiple runs on the full dataset. Second, for clean energy availability, we estimate green energy availability by relying on a forecasting strategy. Finally, we leverage the idea of data stratification for characterizing the payload distribution and to facilitate a partitioning strategy that accounts for both energy as well as processing heterogeneity.

Using the estimates derived (as above) our proposed solution casts the partitioning problem as a multi-objective optimization. The solution to this problem is then used to automatically devise an appropriate partitioning strategy in the presence of environment heterogeneity. The key components of our partitioning framework (Figure 5.1) are: i) a task-specific computational heterogeneity estimator; ii) available green energy estimator; iii) data stratifier (for payload characterization); iv) a Pareto-optimal modeler and v) a data partitioner.

5.3.1 Task-specific Heterogeneity Estimator (I)

As described earlier, in a heterogeneous environment, the time taken by a machine for a specific task depends on the available resources of that machine (speed, time-share,

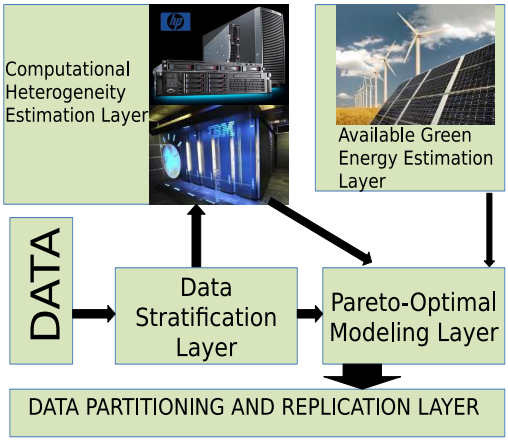


Figure 5.1: The entire framework

and memory capacities), the underpinning complexity of the task as well as the dataset characteristics (size, statistical distribution of payload). In order to model the task specific heterogeneity, we derive a utility function **f** for execution time that accounts for task complexity and available machine resources. Given a sample input payload, an algorithm and a machine, this function can estimate the execution time of that specific task on that payload in the given machine.

We learn the utility function for time **f** by adapting progressive sampling [119] as follows. We take multiple samples of the input data while increasing sample size from 0.05% up to 2% of the data, and run the actual algorithm on these samples and note the execution time for each run on each node in the system. From these (sample size, execution time) pairs, we fit a linear regression model for predicting runtime of the algorithm on any input size. We discuss this choice in more detail in section 5.3.4. We learn a regression model specific to each node in the cluster. This accounts for the difference in terms of machine

speed heterogeneity as we now have execution time models for each machine. This is better than using the stipulated machine CPU speed in three aspects:

1. The CPU speed does not always reflect the true processing rate of an algorithm as there are other factors impacting the speed of an algorithm such as amount of IO required, cache miss pattern, memory access pattern and so on.

2. In virtualized environments, multiple virtual servers are co-located on the same physical host. Two virtual servers with exactly the same configurations could still exhibit different processing rates. One possible reason could be that one of the virtual servers is located on a physical machine which has extremely high load at that instance of time. So the utility function **f** cannot be static, and it has to be learned dynamically.

3. The processing time also depends on the distribution of the data for most data mining algorithms, and the CPU speed cannot capture that aspect.

Our regression model evidently solves problems 1 and 2, as we are running the actual algorithm on the samples, the model learned takes into account the factors such as CPU to IO ratio, cache, and memory access patterns and so on. This model can also solve the problem described in the 3rd point, i.e. data distribution is also a factor in determining runtime, if we can guarantee that the samples generated during progressive sampling phase are representative of the final data partitions on which the algorithm will run on. We can do this by the stratification process which is described in section 5.3.3.

The total execution time for a particular algorithm on node i will be $f(x_i) = m_i x_i + c_i$, where $m_i,\ c_i$ are the learned regression coefficients for node i and x_i is the number of data elements on node i.

5.3.2 Available Green Energy Estimator (II)

In order to account for the dirty energy footprint across individual machines, we need to predict the amount of renewable energy available to each machine. Hence we need a utility function **GE**, which can predict the amount of renewable energy available to a machine over a time interval. In future, we expect such information will be provided by the data center service provider in terms of carbon ratio guarantee or carbon budget. In the current context, we can model the renewable energy availability in a manner similar to Goiri et al. [68]. According to this model, available renewable energy at hour t is,

$$GE(t) = p(w(t))B(t)$$

where $B(t)$ is renewable energy available under ideal sunny conditions, $w(t)$ is the cloud cover and $p(x)$ is attenuation factor. $p(x)$ and $B(t)$ are learned from historical data and $w(t)$ is available from any weather forecast service. To compute availability over a interval, one can sum the GE function over that interval.

Concretely, for the purposes of this work, we leverage the PVWATTS simulator [2] to obtain energy traces for different geographic locations at different points in time. The simulator takes as input the specifications of the solar panel and the location of the solar panel, and based on NREL's weather database and weather models, it outputs the renewable energy production. Though the simulator provides per hour average, one can rescale it to per second average for greater precision. Again we will learn separate models for nodes based on which geographical region the node is from. So for dirty energy footprint of a node for hour t, we can use $g(x_i) = E_i f(x_i) - \sum_{t=1}^{f(x_i)} GE_i(t)$, where E_i is the total energy consumption rate of node i, x_i is the number of data elements in node i and $GE_i(t)$ is the predicted green energy for the hour t for node i.

5.3.3 Data stratifier (III)

Stratification is the process of dividing the data objects into multiple mutually exclusive groups, where each group is called a stratum, and such a stratum exhibit homogeneity. The job of the stratifier hence is to cluster the input data (payload) into a set of strata where each stratum consists of similar data elements . Such a stratification can then be leveraged by our modeler, in conjunction with estimates of computational heterogeneity and green energy availability, to produce a Pareto-optimal partitioning strategy.

The challenge in the stratification step is to do so efficiently in the presence of complex data elements of varying lengths or dimensions (e.g. documents, transactions) while modeling a range of data types from structured (trees, graphs) to unstructured (text). For this purpose, we first sketch the input data to low dimensional sets using a domain specific hash function and then we use a clustering algorithm similar to the Kmodes algorithm to create the strata as outlined previously by Wang et al. [155].

1. Represent the high dimensional data as a set of items. Currently, we support tree, graph and text data. For trees we first represent them using Prufer sequences [124]. We then extract pivots from the Prufer sequences using the least common ancestor relationship in the tree. For example a pivot (a, p, q) would mean node a is the least common ancestor of nodes p and q. Each tree in the input data set is represented as a set of pivots. For graph data sets, we use adjacency list as the pivot set (set of neighbors). For text datasets, we represent each document as a set of words in it. The important thing to note is, *at the end of this step, we have converted our input data type to set data, so now operations can be done in a domain independent way.*

2. The aforementioned sets can still be very high dimensional if the input data is high dimensional. The next step is hence to project the high dimensional sets to a low dimensional space (called sketches) and compute similarity in the low dimensional space that can approximate the similarity of the original sets. We use Jaccard coefficient as a measure of similarity between sets. If x and y are the two sets, the Jaccard similarity is given by:

$$sim(x,y) = \frac{|x \cap y|}{|x \cup y|}$$

We use a locality sensitive hash function call min-wise independent permutations [24] by Broder et. al. to generate the sketches and compute approximate Jaccard similarity very efficiently without much loss in accuracy. Let π be a random permutation of the universal set in question, then the min-wise independent permutation hash function on a data point x is as follows:

$$h_\pi(x) = min(\pi(x))$$

Since the cardinality of the universal set can be extremely large, the above hash function can be very expensive to compute, so we use an approximate algorithm called the min-wise independent linear permutations [22] for computing the hash functions. At the end of this step, *we are left with a sketch of the original input data and the sketch is of orders of magnitude smaller in size that the original input data.* As a result, subsequent operations such as stratification can be done in a very efficient manner.

3. Once compact representation in a low dimensional space is done for all the data points, the final step is to cluster on the sketches to create the strata. We use the compositeKModes clustering algorithm proposed by Wang *et al.* [155] to create the clusters. The standard Kmodes algorithm has the following problem. The cardinality of the universal set is very high and since the small sketches contain very few items, chances of every sketch getting

matched to a cluster center is very low. Consequently, a large number of data points cannot be assigned to any of the clusters because the data points' sketch set has zero-match with the cluster center's sketch set. To overcome this, we use the compositeKModes algorithm, where instead of a cluster center sketch being the mode of each attribute in the feature space, the center sketch maintains L highest frequency elements for each attribute ($L > 1$). In this case, the probability of zero-match decreases significantly as an attribute element in the data point has to match only one of the L values for the same attribute in a center's attribute list. And this variant of Kmodes can also be shown to hold the convergence guarantees of the original Kmodes algorithm. *Using sketch clustering, the input data can be successfully stratified into clusters.*

5.3.4 A Pareto-optimal Model (IV)

Now our goal is to simultaneously minimize the execution time across all partitions, and minimize the sum of the dirty energy consumed by all partitions. Formally, the problem we wish to optimize can be succinctly described as:

$$minimize(v, \sum_{i=1}^{p} g(x_i))$$

$$s.t.\ \forall i,\ v \geq f(x_i), \forall i, x_i \geq 0,\ and\ \sum_{i=1}^{p} x_i = N$$

In the aforementioned formulation v represents the maximum running time across all partitions and $\sum_{i=1}^{p} g(x_i)$ is the total dirty energy consumed by all partitions. Hence the above is a multi-objective optimization problem, with the two objective functions being v and $\sum_{i=1}^{p} g(x_i)$. We wish to find the Pareto frontier [62] of solutions. A solution is a Pareto efficient or optimal one [62] if none of the objectives can be improved upon without degrading at least one. A Pareto frontier is a set of all Pareto efficient solutions. More formally, in terms of our formulation, let v_{xp} and $\sum_{i=1}^{p} g(xp_i)$ be the values of the

objective functions at a solution vector $\vec{xp}$. For this solution to be a Pareto optimal one, it must satisfy following condition: for any other solution vector xq, either $v_{xq} \geq v_{xp}$ or $\sum_{i=1}^{p} g(xq_i) \geq \sum_{i=1}^{p} g(xp_i)$.

We solve the multi-objective optimization problem using a technique called scalarization [77]. Here a single objective function is formed by taking the weighted mean of the multiple objectives. Then any single objective optimization technique can be applied. It can be proved that the solution vector we get by the scalarization technique is a Pareto optimal one [77]. By applying scalarization our problem formulation becomes,

$$minimize(\alpha v + (1-\alpha)\sum_{i=1}^{p} g(x_i))$$

$$s.t.\ 0 \leq \alpha \leq 1, \forall i,\ v \geq f(x_i), \forall i, x_i \geq 0,\ and\ \sum_{i=1}^{p} x_i = N$$

α is the weight factor which controls the tradeoff between the objectives execution time and dirty energy consumption. If we approximate the value of function GE_i to be the mean renewable energy availability rate over a certain period of time that includes the job execution time, then the above formulation becomes a linear programming problem. If the mean renewable energy availability rate is $\bar{GE}$, then for each node i, the factor $E_i f(x_i) - \sum_{t=1}^{f(x_i)} GE_i(t)$ becomes, $E_i f(x_i) - \bar{GE}_i f(x_i) = k_i f(x_i)$, where k_i is a node specific constant. Then our formulation can be further simplified to:

$$\begin{aligned}\alpha v + (1-\alpha)\sum_{i=1}^{p} g(x_i) &= \alpha v + (1-\alpha)\sum_{i=1}^{p} k_i f_i(x_i) \\ &= \alpha v + (1-\alpha)\sum_{i=1}^{p} k_i(m_i x_i + c_i)\end{aligned}$$

The described formulation is a linear programming problem and hence can be efficiently solved. The solution to this linear programming problem always results in a Pareto-optimal

solution, i.e. a change in the solution vector will degrade at least one of the objective functions. Specifically, optimality is guaranteed when the execution time is approximately linearly related to the data size and the fluctuations in the renewable energy availability are minimal so that the availability is close to the mean energy supply. Empirically, even when these conditions are not satisfied, this model will perform better than partitioning naively with equal sized partitions as we shall shortly demonstrate. Though in that case, a better solution will exist.

Another option we considered and empirically evaluated, is to fit a more general functional form to the utility functions, such as higher order polynomials for the regression model. Theoretically, this makes sense as any arbitrary function can be approximated by polynomials using the Taylor approximation. But practically it is not a feasible option as such models will take a very high number of samples to fit the curve properly. Too few points will invariably over fit the points to the model. And we cannot afford too many samples in our progressive sampling step as collecting a sample implies running the actual algorithm on a small sample of the data. Under these circumstances the linear regression model was found to be quite effective on all configurations we evaluated and is moreover easily trained with very few samples and the resulting formulation leads to a linear programming problem, that can be efficiently solved.

Usually, for a multi-objective optimization, there is a set of Pareto optimal solutions. This set is known as the Pareto frontier. Our formulation, which is a weighted mean of the individual objectives, generates only one point on the Pareto frontier. The user-defined parameter α controls which point we get in the Pareto frontier. Hence, setting α to a high value will imply a partitioning scheme where time will be improved more than energy and setting α to a lower value will optimize the energy function better. In fact, our Het-Aware

scheme is a special case where α is set to 1.0. The system then will only optimize for time. Note that selecting α can be challenging as the scale of the two objective functions (time and energy) are different. The energy function has a much higher scale than the time function. This implies, to focus more on optimizing time than energy, we have to set a very high value of α (α is the coefficient of the time function). We believe in future this problem can be avoided by normalizing both the objective functions to 0-1 scale, and then both functions will be equally sensitive to changes in α.

5.3.5 Data Partitioner (V)

The final component of our framework is the data partitioner. This component is responsible for putting the final data partitions into the machines based on the output of the modeling step. Currently, we support the final partitions to be data partitions stored on disk or data partitions stored on Redis NoSQL store. In future, we plan to support NoSQL stores that guarantees SLOs while reducing cost [144]. After the optimization step is done, the framework already knows how many data items to put in each partition. Our data partitioner supports two types of partitions. Both of the schemes are driven by the stratification process.

- **Making each partition representative of payload:** The goal here is to make each partition a representative of the entire data. Such a representative partitioning can be achieved by making each partition a stratified sample without replacement of the data. Cochran [42] showed that a stratified sample approximates the underlying true data distribution much better than a simple random sample. As a result, our partitions will be good representatives of the global data, especially if the data has a large number of strata and each partition is relatively small with respect to the total input data. Since our stratification step already creates the strata, we can proportionally allocate elements to each stratum to create the required partitions.

- **Placing similar elements together:** The goal here is to group similar types of data items together. Again we can achieve such a partitioning scheme by using the strata created by our stratification process. Ideally, we would like to make individual stratum a partition by itself. This would ensure minimum entropy for all partitions. But we have to take to into account the constraint set by the optimizer, it has already decided what the partition sizes are, to optimally load balance. And usually, the number of strata are much higher than the number of partitions. In order to do the partitioning, in this case, we first order the elements one according to the strata id, i.e. all elements of strata 1 followed by elements of strata 2 and so on. Once this ordering is created, we create the partitions by taking chunks of respective partition sizes from this ordered data.

  Note that, samples of both kinds of partitions can be generated by the stratifier as only the data clusters are required. Those are the samples which the stratifier feeds to the heterogeneity estimator so that in the progressive sampling step, the samples are representative of the final partition payload. This makes the heterogeneity estimator aware of the payload characteristics.

5.4 Implementation

Our data partitioning framework is implemented in C and C++. We use the C library API of Redis NoQSL store as our underlying storage medium. Note that we do not use the cluster mode of Redis as in that mode we do not have control over which key goes to which partition and our whole idea relies on the fact that we will be able to place data items according to our stratification and optimization rules. Hence, we run one instance of Redis server in each of our cluster nodes, and manually manage communication from the framework middleware level. We need a global barrier module for our framework as

the pivot extraction, sketch generation, sketch clustering and final data partitioning have to be separated by barriers. We used the atomic *fetch-and-increment* command provided by Redis to create a global barrier routine. Since there could be millions of data items, each of which can be a high dimensional set, storing them could imply millions of get/put requests in Redis and many of those requests could be to remote machines. This can evidently cause a huge performance hit. We use a storage data structure which can avoid this excessive get/put requests on Redis. Instead of storing the individual attribute values of a data item, we store the item as a sequence of raw bytes and we maintain a list of such sequences for a list of data items. The first four bytes in the sequence contain the length of the data object. Note we use the Redis list structure here. This gives us the freedom to access the entire data set of a partition in a single get/put operation, and the access to individual data items from a get/put request as well. To further improve batching of requests we use the pipelining feature of Redis, where requests are batched up to the preset pipeline width and then sent out. In Redis, this is known to substantially improve the response times.

There are three tasks in our framework which are done in a centralized fashion - the global barrier routine, the clustering and creation of the representative data sample which every node will run on to get runtime and energy consumption estimates. Note that we chose to do the clustering in a centralized manner as the compositeKmodes algorithm is run on the sketches rather than the actual data. The size of the sketches of a dataset is of orders of magnitude smaller than the raw data size, which is why it is easy to fit in a single machine. As a result, the clustering can run with zero communication overhead. We saw that doing the clustering in distributed fashion over the sketches is prohibitive in terms of runtime. Even though the two tasks have to be managed by a master node, they need not be the same node. In other words, we choose two separate nodes in the cluster for the two tasks.

This gives us some level of decentralization and better load balancing. We also choose type 1 nodes (fastest) as the master nodes if available. If not, then we select type 2, type 3 or type 4 in that order of priority.

5.5 Experimental evaluation

In this section, we seek to examine the efficacy of the proposed Pareto framework for analytic workloads on heterogeneous systems. Our workloads are drawn from those commonly used in the search, news and social media industry and include both the analysis of structured (e.g. graph) and unstructured (e.g. text) data. Specifically, we use two types of distributed workloads: (i) frequent pattern mining - a compute intensive workload, where even if the input data size is small, the intermediate data can be huge and (ii) compression - a data intensive workload that usually runs on huge quantities of data. We seek to answer the following questions:

- Is there a tangible benefit to heterogeneity-aware partitioning for such workloads? For both unstructured and structured data workloads?
- How effective is the Pareto-optimal model? Does using different values of the α result in interesting differentials in runtime and dirty energy usage?

5.5.1 Setup

We use a cluster of machines to run our experiments. The individual nodes consist of 12 cores with Intel Xeon 2.2GHz frequency. Each machine has a RAM of 48GB. Since the experiments we run are cluster heterogeneity aware, and the machines in this cluster are homogeneous, we need to introduce heterogeneity both in terms of speed and renewable energy availability. We introduce heterogeneity in the following way:

1. We use 4 different types of machine speeds in our experiments. The idea is to use machines with relative speeds $x, 2x, 3x$ and $4x$. The way we do it is by introducing busy loops in the homogeneous cluster. Since there are 12 cores per machine, type 1 nodes have no busy loops, type 2 nodes will have 12 busy loops, type 3 nodes will have 24 busy loops and type 4 nodes will have 36 busy loops running in parallel.

2. We use the NREL simulator [2] to simulate renewable energy heterogeneity. Again we introduce 4 types of nodes. We select 4 of Google's data center locations and create renewable energy traces for those locations from the weather database and models of the PVWATTS simulator. We found out the server power consumption of each machine from HP SL server specifications (1200 WATTS 12 cores). We used the individual processor power consumption value from Intel Xeon (95 Watts), which implies base operating power is (1200 - 95*12) = 60 Watts. We then generated the 4 types of machines by running 0, 12, 24, 36 busy while loops in the 4 machines as described above. And we assumed that the fastest machine has 4 cores, 2nd fastest 3 cores, then 2 cores and the slowest one with 1 core. Therefore, the power consumption for the 4 types of machines is $60+4*95 = 440$ Watts, $60+3*95 = 345$ Watts, $60+2*95 = 250$ Watts and $60+1*95 = 155$ Watts respectively.

5.5.2 Datasets

We use 5 real world datasets from 3 different domains namely - graphs, trees, and text datasets. Our datasets vary from 50,000 trees set to 15 million nodes graph and since they are collected from different domains, the underlying data distributions and characteristics will largely vary. Table 5.1 contains the description of the datasets. The datasets are collected from [1, 3, 97].

Dataset	Type	Size
SwissProt	Tree	# of trees - 59545, Nodes - 2977031
Treebank	Tree	# of trees - 56479, Nodes - 2437666
UK	Graph	Nodes - 11081977, Edges - 287005814
Arabic	Graph	Nodes - 15957985, Edges - 633195804
RCV1	Text	# of docs - 804414, vocab size - 47236

Table 5.1: Datasets

5.5.3 Applications and Results

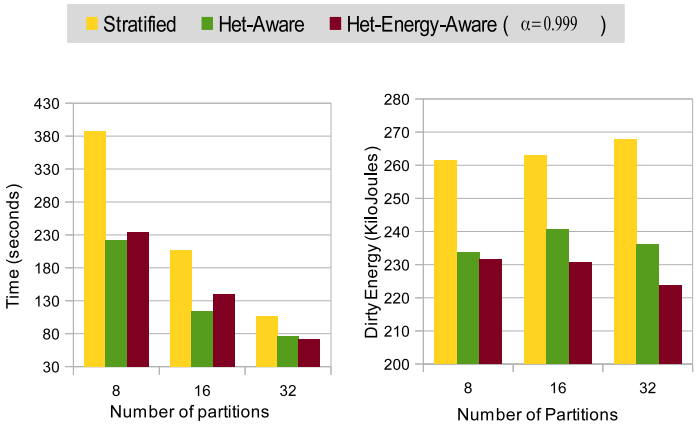
5.5.3.1 Frequent Pattern Mining

Frequent pattern mining has been one of the most common data mining applications. The goal of the frequent pattern mining problem is to find the frequent co-occurring items in the entire data set. The co-occurring items have to be present in at least a certain percentage (called support) of the entire dataset. There has been a lot of research in developing fast algorithms for frequent pattern mining on text or transactional data [4, 167], trees [149] and graphs [163]. Since the number of candidate patterns to check against a support can at worst be exponential, frequent pattern mining algorithms can be really slow. Consequently distributed versions of these algorithms are of utmost importance. We use the partition based distributed frequent pattern mining algorithm proposed by Savasere et. al. [132]. The algorithm works by first finding the locally frequent patterns in each partition and then a global scan is required to prune out the false positive patterns. If each partition has similar number of candidate patterns to evaluate, then depending on the system heterogeneity, faster machines will finish processing faster, however, the overall execution time will be bottlenecked by the slow running partitions. Similar is the case with energy heterogeneity. The partitioning scheme should try to schedule more computation to the machines which

have higher availability of renewable energy. Additionally, a naive partitioning scheme may result in substantial skew in the number of candidate patterns to process in each partition. The execution time for the entire job will increase even if a single partition generates too many patterns. Hence to provide a fully heterogeneity aware partitioning, the partitions should be homogeneous in terms of the payload characteristics as well. We achieve this by our stratified partitioning strategy which tries to make each partition a representative of the payload.

We evaluate performance in terms of execution time and total dirty energy consumed across all machines. We report results under three different partitioning strategies - 1) Stratified partitioning, 2) Het-Aware ($\alpha = 1.0$) stratified partitioning and 3) Het-Energy-Aware stratified partitioning. A simple random partitioning strategy performs much worse than our baseline (stratified strategy) [54, 155]. For the Het-Energy-Aware scheme, we set the parameter α to 0.999. We will show that by controlling α, we are able to find a solution that simultaneously beats the execution time and the dirty energy footprint of the baseline strategies which create payload partitions of equal sizes. α needs to be set to high values (close to 1.0) to find decent tradeoffs between time and energy as the two objective functions have different scales. In future we plan to normalize both objectives to 0-1 range. We ran 2 variants of the frequent pattern mining algorithms:

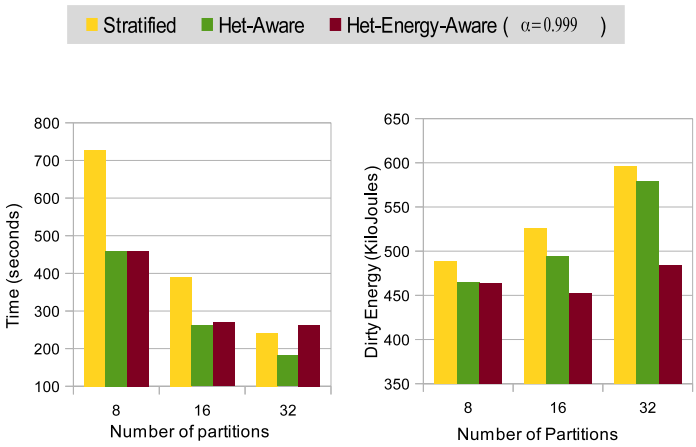
Frequent Tree Mining: We ran the frequent tree mining algorithm [149] on our 2 tree datasets. Figure 5.2 reports the execution time and dirty energy consumption on the Swiss protein dataset and the Treebank dataset. Results indicate that using our Het-Aware strategy can improve the runtime by 43% for the 8-partition configuration for Treebank (Figure 5.2c), and this is the best strategy when only execution time is of concern. However, Figures 5.2d



(a) Swiss (b) Swiss

(c) Treebank (d) Treebank

Figure 5.2: Frequent Tree Mining on Swiss Protein and Treebank Dataset



(a) Execution Time (b) Dirty Energy

Figure 5.3: Frequent Text Mining on RCV1 corpus

and 5.2b , that have the energy consumption numbers of these strategies, show that Het-Aware solution is not the most efficient one in terms of dirty energy consumption. Here the Het-Energy-Aware scheme performs the best. In the same 8-partition configuration as described before, the Het-Energy-Aware strategy reduces the execution time by 36% while simultaneously reducing the dirty energy consumption by 9%.

Text Mining: We run the apriori [4] frequent pattern mining algorithm on the RCV corpus. The execution time numbers are reported in Figure 5.3a. Again the Het-Aware scheme is the best with improvement up to 37% reduction in execution time over the stratified partitioning strategy with 8 partitions. The energy numbers are provided in Figure 5.3b. The Het-Energy-Aware scheme for the 16-partition configuration reduced the as expected reduced the runtime by 31%, while consuming 14% less energy than the stratified partitioning scheme.

5.5.3.2 Graph compression

We test our partitioning scheme on distributed graph compression algorithms. The idea is, we split the input data into p partitions. And then we compress the data in individual partitions independently. We use two compression algorithms LZ77 [175] and webgraph [23] to compress the data of individual partitions.

Here we benefit from the partitioning strategy which tries to group similar elements together in a single partition. If a partition comprises of elements which are very similar, then a partition can be represented by a small number of bits. By creating such low entropy partitions, one can get very high compression ratio.

We evaluate performance by the execution time. Again we compare three strategies, stratified partitioning with no heterogeneity awareness, Het-Aware stratified partitioning and Het-Energy-Aware stratified partitioning. Here we set the parameter α to be 0.995 instead of 0.999 as was the case in the frequent pattern mining experiments. Due to reasons explained in Section 5.3.4 the execution time should deteriorate quite a bit and should be close to the baselines, while dirty energy consumption rate should improve significantly.

Figure 5.4 reports the performance numbers (execution time and dirty energy consumption) as well the quality (compression ratio) on both the UK dataset and the Arabic dataset. Our Het-Aware strategy improves the execution time by 51% on the Arabic dataset for the 8-partition configuration (Figure 5.4c). Our Het-Energy-Aware scheme reduces the execution time by only 9%, but simultaneously it reduces the dirty energy consumption by 26% on the same configuration as described above. This also shows the impact of setting a lower α than the frequent pattern mining experiments. The execution time improvements have gone down and dirty energy consumption rate has improved substantially.

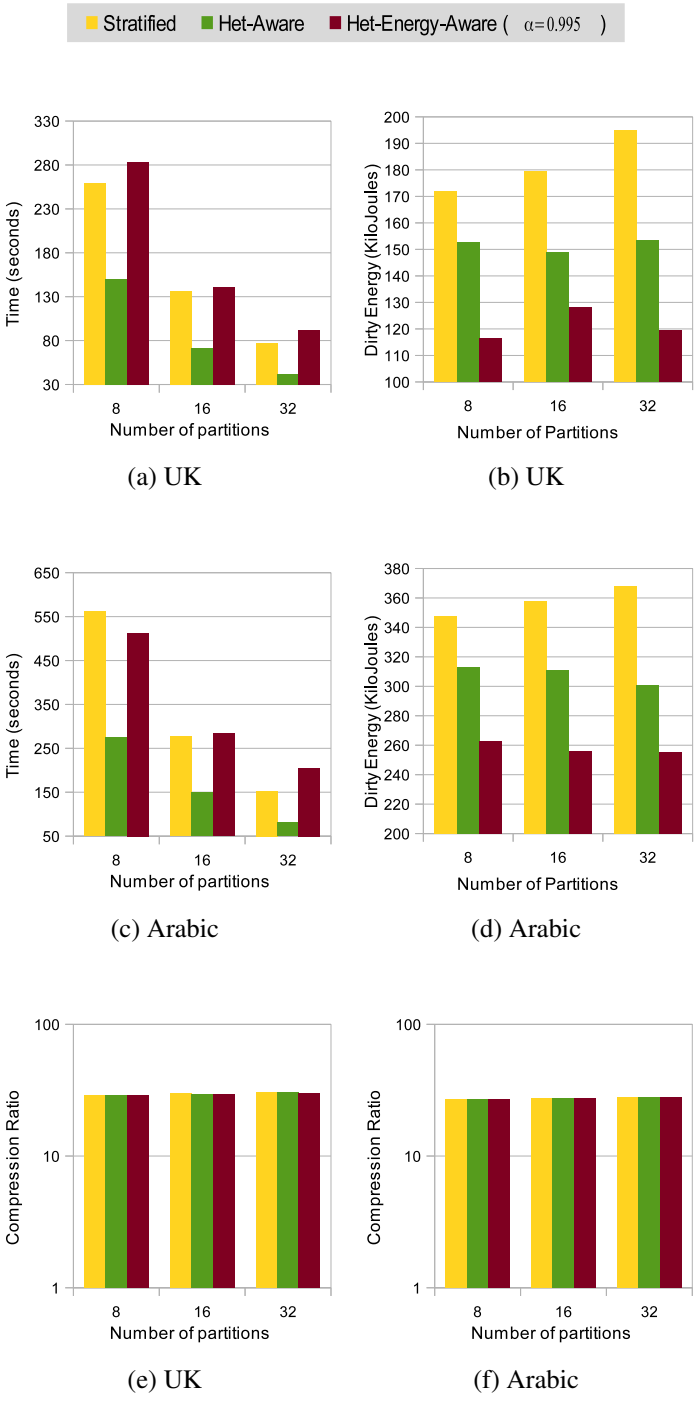


Figure 5.4: Graph compression results on UK and Arabic webgraphs

Strategy	Time (seconds)	Compression ratio
Stratified	18	18.33
Het-Aware	11	18.2
Het-Energy-Aware	12	18.01

Table 5.2: LZ77 compression on UK dataset with 8 partitions

Strategy	Time (seconds)	Compression ratio
Stratified	38	18.3
Het-Aware	35	18.26
Het-Energy-Aware	40	18.14

Table 5.3: LZ77 compression on Arabic dataset with 8 partitions

We evaluate the quality of our partitioning schemes by comparing the compression ratios achieved by each scheme. Our heterogeneity aware stratified schemes match the compression ratio of the baseline stratified scheme. Hence we are able to vary the partition sizes to account for better load balancing without any degradation of quality. The technique of reordering the data points according to clusters and creating chunks of variable sizes is able to generate low entropy partitions.

We also run experiments with the very common LZ77 compression algorithm. Tables 5.2 and 5.3 report the performance and quality numbers for the UK and the Arabic datasets respectively for the 8-partition setting. Lz77 is extremely fast, so there are no gains from our heterogeneity aware schemes. The compressibility of our heterogeneity aware techniques is comparable to that of the stratified strategy.

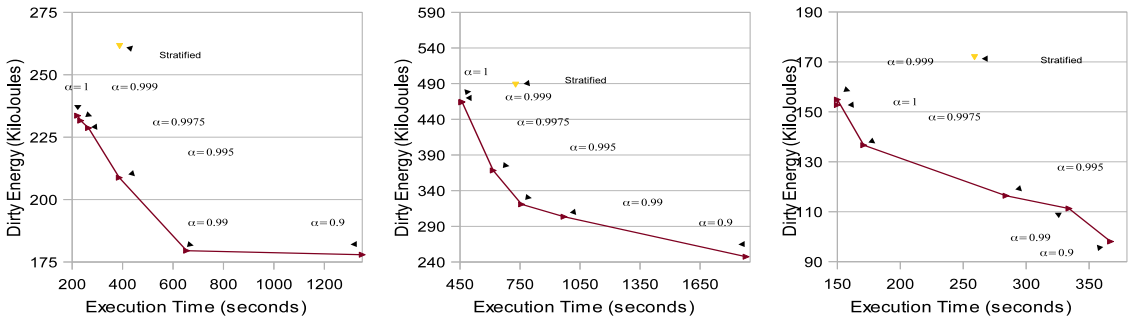
5.5.4 Understanding the Pareto-Frontier:

Here we study the effect parameter α has on time-energy tradeoff curve (Pareto-frontier) for all three workloads we consider. For all workloads (Figure 5.5) we vary the value of α from 1 to 0 and study the impact on execution time and dirty energy consumption (for 8 partitions). There are two major trends one observes.

First, it is clear that by changing the value of α focus can be effectively shifted from execution time minimization to dirty energy minimization. The magenta line shows this shift. At $\alpha = 1.0$ (extreme left point) execution time is minimum, while dirty energy consumption is maximum for all workloads. This point also represents the Heterogeneity- aware scheme reported earlier. As α is reduced the runtime increases but the dirty energy consumed is reduced. We note that at an α value of about 0.9 dirty energy is typically minimized but at this point the execution time is fairly high. The rationale is that most of the load is placed on the node that harnesses the most green energy leading to severe load imbalance. In other words at this point the optimizer puts almost all of the payload in the machine with lowest dirty energy footprint. Further lowering α does not have any additional impact.

Second, we observe that the baseline strategy of stratified partitioning is significantly above and to the right of magenta line (yellow points). Consequently, a simple stratified strategy results in sub-optimal a solution (not Pareto-efficient).

Third, in Figure 5.6 we evaluated whether our methodology is able to generalize over different parametric settings on the same dataset. We changed the support threshold (the key parameter for frequent pattern mining) for both tree and text datasets and plotted the Pareto frontiers by varying α as described before. For both the datasets, we clearly see that our method is able to find the Pareto frontiers nicely. Hence our framework can tradeoff of between performance and dirty energy across different parametric settings of the same

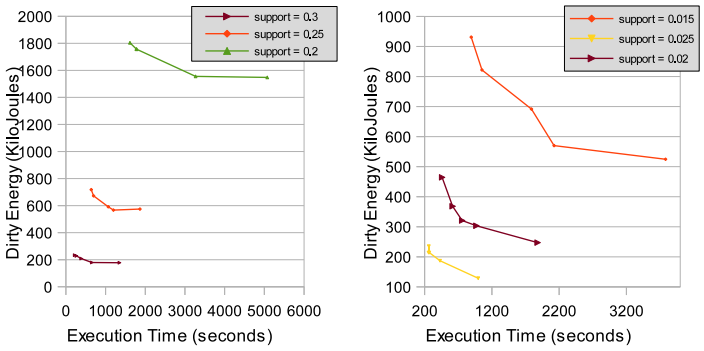


(a) Swiss dataset (b) RCV dataset (c) UK dataset

Figure 5.5: Pareto frontiers on a) Tree, b)Text, and c) Graph workloads (8 partitions). Magenta arrowheads represent Pareto-frontier (computed by varying α). Note that both baselines: Stratified (yellow inverted arrowhead); lie above the Pareto frontier (not Pareto-efficient) for all workloads.

workload. This is particularly important in the context of frequent pattern mining as support is an intrinsic property of the dataset – to find interesting patterns in different datasets, the support has to be adjusted accordingly.

To summarize it is clear that accounting for payload-distribution can result in significant performance and energy gains. Coupled with heterogeneity- and green-aware estimates these gains can be magnified.



(a) Swiss dataset (b) RCV dataset

Figure 5.6: Pareto frontiers on a) Tree and b)Text (8 partitions) by changing the support thresholds.

5.6 Related Works

Data partitioning and placement is a key component in distributed analytics. Capturing of representative samples using the stratified sampling technique on large scale social networks using MapReduce has been investigated by Levin et. al. [96]. Meng [108] developed a general framework for generating stratified samples from extremely large scale data (need not be social network) using MapReduce. Both of these techniques are effective for creating a single representative sample, however, our goal in this work is to partition the data such that each partition is statistically alike. Duong et. al. [54] develops a sharding (partitioning) technique for social networks that performs better than random partitioning. This technique utilizes information specific to social networks to develop effective partitioning strategies. In contrast, our goal is to develop a general framework for data partitioning in the context of distributed analytics. Another related work by Wang et. al. [155] provides a method to mitigate data skew across partitions in the homogeneous context. In this work we propose

to design a framework for the heterogeneous context where the heterogeneity is in terms of processing capacity and green energy availability across machines. Performance aware and energy aware frameworks are studied extensively in the context of cloud and database workloads [40, 87, 94, 121, 125, 154]. However, these techniques are not payload aware which is extremely critical for large scale analytics workloads. Along with performance and energy skew, data skew also plays a significant role in the performance of analytics tasks.

5.7 Concluding Remarks

The key insight we present is that both the quality and performance (execution time and dirty energy footprint) of distributed analytics algorithms can be affected by the underlying distribution of the data (payload). Furthermore, optimizing for either execution time or minimizing dirty energy consumption leads to a Pareto-optimal tradeoff in modern heterogeneous data centers. We propose a heterogeneity-aware partitioning framework that is conscious of the data distribution through a lightweight stratification step. Our partitioning scheme leverages an optimizer to decide what data items to put it which partition so as to preserve the data characteristics of each partition while accounting for the inherent heterogeneity in computation and dirty energy consumption. Our framework also allows data center administrators and developers to consider multiple Pareto-optimal solutions by examining only those strategies that lie on the Pareto-frontier. We run our placement algorithm on three different data mining workloads from domains related to trees, graphs and text and show that the performance can be improved up to 31% while simultaneously reducing the dirty energy footprint by 14% over a highly competitive strawman that also leverages stratification.

Chapter 6: Modeling and Managing Latency of Distributed NoSQL stores

In this chapter, we develop a middleware called Zoolander for managing the latency of NoSQL stores in the face of non-deterministic anomalies causing slowdowns or slowdowns due to heterogeneous computing environments. More generally this middleware can provide probabilistic guarantees on the latency of any NoSQL store and hence can be used for any latency sensitive applications such as internet services where the data is distributed and replicated in a cluster of nodes. *For distributed large datasets, fast storage accesses are key to interactive analytics.*

6.1 Introduction

Zoolander masks slow storage accesses via replication for predictability, a historically dumb idea whose time has come [114]. Replication for predictability scales out by copying the exact same data across multiple nodes (each node is called a duplicate), sending all read/write accesses to each duplicate, and using the first result received. Historically, this approach has been dismissed because adding a duplicate does not increase throughput. But duplicates can reduce the chances for a storage access to be delayed by a background job, shrinking heavy tails[1] Very recent work has used replication for predictability but

[1] In this chapter, we use the term *heavy tailed* to describe probability distributions that are skewed relative to normal distributions. Sometimes these distributions are called fat tailed.

only sparingly with ad-hoc goals [7, 46, 152]. Zoolander fully supports replication for predictability at scale.

Zoolander can also scale out by reducing the accesses per node using partitioning and traditional replication. Its policy is to selectively use replication for predictability only when it is the most efficient way to scale out (i.e., it can meet SLO using fewer nodes than the traditional approaches). Zoolander implements this policy via a biased analytic model that predicts service levels for 1) the traditional approaches under ideal conditions and 2) replication for predictability under actual conditions. Specifically, the model assumes that accesses will be evenly divided across nodes (i.e., no hot spots). As a result, the model overestimates performance for traditional approaches. In contrast, our model predicts the performance of replication for predictability precisely, using first principles and measured systems data. Despite its bias, our model provided key insights. First, replication for predictability allows us to support very strict, low latency SLOs that traditional approaches cannot attain. Second, traditional approaches provide efficient scale out when system resources are heavily loaded, but replication for predictability can be the more efficient approach when resources are well provisioned.

We implemented Zoolander as a middleware for existing key-value stores, building on prior designs for high throughput [74, 76, 152]. Zoolander extends these systems with the following features:

1. High throughput and strong SLO for read and write accesses when clients do not share keys. Zoolander also supports shared keys but with lower throughput.

2. Low latency along the shared path to duplicates via reduced TCP handshakes and client-side callbacks.

3. Reuse of existing replicas to reduce bandwidth needs.

4. A framework for fault tolerance and online adaptation.

We used write- and read-only benchmarks to validate Zoolander's analytic model for replication for predictability under scale out. The model predicted actual service levels, i.e., the percentage of access times within SLO latency bounds, within 0.03 percentage points.Replication for predictability increased service levels significantly. On the write-only workload using 4 nodes, Zoolander achieved access times within 15ms with a 4-nines service level (99.991%). Using the same number of nodes, traditional approaches achieved a service level of only 99%—Zoolander increased service levels by 2 orders of magnitude.

We set up Zoolander on 144 EC2 units and issued up to 40M accesses per hour, nearly matching access rates seen by popular e-commerce services [13, 43, 75]. We also varied the access rate in a diurnal pattern [142]. By using both replication for predictability and traditional approaches, Zoolander provided new, cost effective ways to scale. At night time, when arrival rates drop, Zoolander decided not to turn off under used nodes. Instead, it used them to reduce costly SLO violations. Zoolander's approach reduced nightly operating costs by 21%, given cost data from [75, 151]. With better data migration, Zoolander could have reduced costs by 32%.

6.2 Background

6.2.1 Motivation

Internet services built on top of networked storage expect data accesses to complete quickly all of the time. Many companies now include latency clauses in the service level objectives (SLOs) given to storage managers. Such SLOs may read, "98% of all storage accesses should complete within 300ms provided the arrival rate is below 500 accesses

per second [50, 143, 152]." When these SLOs are violated, Internet services become less usable and earn less revenue. Consider e-commerce services. SLO violations delay web page loading times. As a rule of thumb, delays exceeding 100ms decrease total revenue by 1% [129]. Such delays are costly because revenue, which covers salaries, marketing, etc., far exceeds the cost of networked storage. A 1% drop in revenue can cost more than an 11% increase in compute costs [151].

Many networked storage systems meet their SLOs by scaling out, i.e., when access rates increase, they add new nodes. The most widely used scale-out approaches partition or replicate data from old nodes to new nodes and divide storage accesses across the old and new nodes, reducing resource contention and increasing throughput [50, 71, 101]. However, background jobs, e.g., write-buffer dumps, garbage collection, and DNS timeouts, also contend for resources. These periodic events can increase access times by several orders of magnitude.

6.2.2 Related Work

Zoolander improves response times for key value stores by masking outlier access times. Contributions include: 1) a model of replication for predictability that is blended with queuing theory, 2) full, read-and-write support for replication for predictability, and 3) experimental results that show the model's accuracy and cost effective application of replication for predictability. Related work falls into the categories outlined below.

Replication for predictability and cloning: Google's BigTable re-issues storage accesses whenever an initial access times out (e.g., over 10ms) [46,47]. Outliers will rarely incur more than 2 timeouts. This approach applies replication for predictability only on known outliers, reducing its overhead compared to Zoolander. Writes present a challenge for BigTable's

approach. If writes that are not outliers are sent to only 1 node, duplicates diverge. If instead, they are sent to all nodes re-issued accesses would not mask delays because they would depend on slow nodes. Zoolander avoids these problems by sending all writes to all replicas.

SCADS revived replication for predictability, noting its benefits for social computing workloads [12]. SCADS sent every read to 2 duplicates [152] and supported read-only or inconsistent workloads. Replication for predictability strengthened service levels by 3–18%. Zoolander extends SCADS by scaling replication for predictability, modeling it, and supporting consistent writes. Empirical evaluation will show that, as arrival rates increase, our model can find replication policies that outperform the fixed 2-duplicate approach.

Data-intensive processing uses cloning to mask outlier tasks. Early Map-Reduce systems cloned tasks when processors idled at the end of a job [48]. Mantri et al. [7] used cloning throughout the life of a job to guard against failures. In both cases, the number of duplicates was limited. Also, map tasks issue only read accesses. Recent work used cloning to mask delays caused by outlier map tasks [6], providing a topology-aware adaptive approach to save network bandwidth. Like Zoolander, this work focused on cost effective cloning. Zoolander's model advances this work, allowing managers to understand the effect of budget policies in advance. Another recent work [86] sped up data-intensive computing via replication for predictability. This work defines budgets in terms of reserve capacity and uses recent models on map-reduce performance [172].

Adaptive partitioning and load balancing: Heavy tail access frequencies also degrade SLOs. Hot Spots are shards that are accessed much more often than typical (median) shards. Queuing delays caused by hot spots can cause SLO violations. Further, hot spots may shift between shards over time. SCADS [152] threw hardware at the problem by migrating the hottest keys within a shard via partitioning and replication. Other works have

extended this approach to handled differentiated levels of service [140] and also for disk based systems [110]. Consistent hashing provides probabilistic guarantees on avoiding hot spots [145, 173]. [78] extends consistent hashing by wisely placing data for low cost migration in the event that a hot spot arises. Locality aware placement can also reduce the impact of hot spots [90].

Both replication for predictability and power-of-two load balancing [113] involve sending redundant messages to nodes. However, in load balancing, the nodes do not share a consistent view of data. Just-idle-queue load balancing includes a related sub problem where an idle node must update exactly 1 of many queues [104]. Here, taking the smallest queue is like taking the fastest response in replication for predictability and reduces heavy tails.

Removing performance anomalies: Background jobs are not the only root cause of heavy tails, performance bugs that manifest under rare runtime conditions also degrade response times. Removing performance bugs requires tedious and persistent effort. Recent research has tried to automate the process. Shen et al. use "reference executions" to find low level metrics affected by bug manifestations, e.g., system call frequency or pthread events [138]. These metrics uncovered bugs in the Linux kernel. Attariyan et al. used dynamic instrumentation to find bugs whose manifestation depended on configuration files [14]. Recent works have found bugs at the application level [88, 166]. Debugging performance bugs and masking their effects, as Zoolander does, are both valuable approaches to make systems more predictable, but neither is sufficient. Some root causes, like cache misses [13], should be debugged. Whereas, other root causes manifest sporadically but, if they were fixed, could unmask bigger problems [143].

The operating system and its scheduler are a major reason for heavy tails. Two recent studies reworked memcached, removing the operating system from the data path via

RDMA [85, 146]. While many companies can not run applications like memcached outside of kernel protection, these studies suggest that the OS should be redesigned to reduce access-time tails.

6.3 Replication for Predictability

Traditional approaches to scale out networked storage share a common goal: They try to reduce accesses per node by adding nodes. While such approaches improve throughput, there is a downside. By sending each access to only 1 node, there is a chance that accesses will be delayed by background jobs on the node [46]. Normally, background jobs do not affect access times, but when they do interfere, they can cause large slowdowns. Consider write buffer flushing in Cassandra [74]. By default, writes are committed to disk every 10 seconds by flushing an in-memory cache. The cache ensures that most writes proceed at full speed without incurring delay due to a disk access. However, if writes arrive randomly and buffer flushes take 50ms, we would expect buffer flushes to slow down 0.5% of write accesses ($\frac{50ms}{10s}$).

Figure 6.1 compares replication for predictability against traditional, divide-the-work replication. The latter processes each request on one node. When a buffer flush occurs, pending accesses must wait, possibly for a long time. However, by sending all accesses to N nodes and taking the result from the fastest, replication for predictability can mask $N-1$ slow accesses, albeit without scaling throughput. In this section, we generalize this example by modeling replication for predictability. Our analytic model outputs the expected number of storage accesses that complete within a latency bound. It allows us to compare replication for predictability to traditional approaches in terms of SLO achieved and cost.

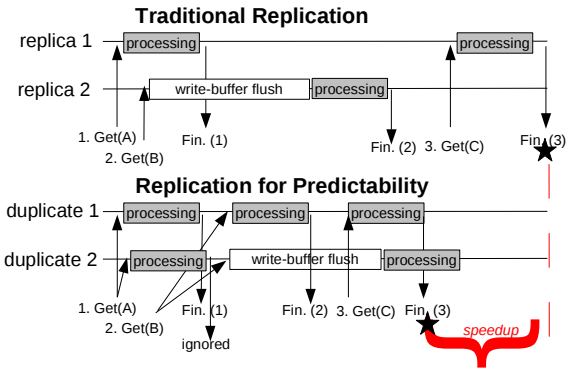


Figure 6.1: Replication for predictability versus traditional replication. Horizontal lines reflect each node's local time. Numbered commands reflect storage accesses. Get #3 depends on #1 and #2. Star reflects the client's perceived access time.

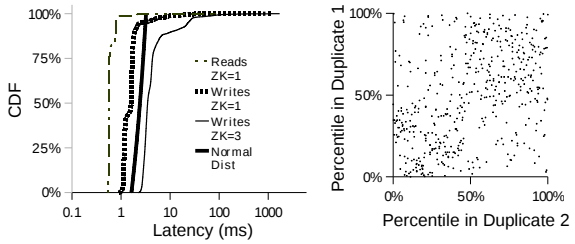
6.3.1 First Principles

Our model is based on the following first principles:

1. Outlier access times are heavy tailed. Background jobs can cause long delays, producing outliers that are slower and more frequent than Normal tails.

2. Outliers are non-deterministic with respect to duplicates. To mask outliers, slow accesses on 1 duplicate can not spread to others. Replication for predictability does not mask outliers caused by deterministic factors, e.g., hot spots, convoy effects, and poor workload locality.

To validate our first principles, we studied storage access times in our own local, private cloud. We use a 112 node cluster, where each node is a core with at least 2.4 GHz, 3MB L2 cache, 2GB of DRAM memory, and 100GB of secondary storage. Our virtualization software is User-Mode Linux (UML) [53], a port of the Linux operating system that runs in user space of any X86 Linux system. Thus, RedHat Linux (kernel 2.6.18) serves as our VMM. Custom PERL scripts designed in the mold of Usher [107] allow us to 1) run preset virtual machines on server hardware, 2) stop virtual machines, 3) create private networks,



(a) (b)

Figure 6.2: Validation of our first principles. (A) Access times for Zookeeper under read- and write-only workloads exhibit heavy tails. (B) Outlier accesses on one duplicate are not always outliers on the other.

and 4) expose public IPs. Our cloud infrastructure is compatible with any public cloud that hosts X86 Linux instances. Later in this chapter, we will scale out on Amazon EC2.

We set up Zookeeper [76] and performed $100,000$ data accesses one after another. Zookeeper is a key-value store that is widely used to synchronize distributed systems. It is deployed as a cluster with ZK nodes. Writes are seen by $\frac{ZK}{2}+1$ nodes. Reads are processed by only 1 node. Figure 6.2a plots the cumulative distribution function (CDF) for Zookeeper under read-only and write-only workloads. The coefficient of variation ($\frac{\sigma}{|\mu|}$), or COV, shows the normalized variation in a distribution. Generally, COV equal or below 1 is considered low variance. We compared the plots in Figure 6.2a by 1) computing COV before the tail, i.e., up to the 70^{th} percentile and 2) computing COV across the whole CDF. Before the tail, COV was below 1. Across the entire distribution, COV was much higher, ranging from 1.5–8.

To visually highlight the heaviness of the tails, Figure 6.2a also plots a normal distribution with standard deviation and mean that were 25% larger than 90% of write times in ZK=1. Note, COV in a normal distribution is 1. The tails for both reads and writes under ZK=1 overtake the normal distribution, even though the normal distribution has a larger mean.

We also found that tails became heavier as complexity increased. Writes in a single-node Zookeeper led to local disk accesses that didn't happen under reads. Writes in 3-node Zookeeper groups send network messages for consistency.

We can also interpret each (x, y) point in Figure 6.2a as a latency bound and an achieved service level. If access times followed a normal distribution, a latency bound that was 3 times the mean would provide a service level of 99.8%. Figure 6.2a shows that Zookeeper's service levels were only 98.8% of reads,96.0% of 1-node writes, and 91.5% of 3-node writes under that latency bound. To support a strict SLO that could cover 99.99% of data accesses, the latency bound would have risen to 16X, 26X, and 99X relative to the means.

Heavy tails affect many key value stores, not just Zookeeper. Internal data from Google shows that a service level of 99.9% in a default, read-only BigTable setup would require a latency bound that is 31X larger than the mean [46]. Others have noticed similar results on production systems [15, 75]. We also measured read access times in a single Memcached node, a key-value widely used in practice and in emerging sustainable systems [13, 136]. We saw a coefficient of variation of 1.9, and, under a lax latency bound, only a 98.3% service level was achieved. Finally, we ran the same test with Cassandra [74], another widely used key-value store, deployed on large EC2 instances. The coefficient of variation was 6.4.

Figure 6.2b highlights principle #2. Across two Zookeeper runs that receive the same requests under no concurrency, we show the percentile of each storage access. If slow service times were workload dependent, either the bottom right or upper left quartiles of this plot would have been empty, i.e., slow accesses on the first run would be slow again on the second. Instead, every quartile was touched.

6.3.2 Analytic Model

This subsection references the symbols defined in Table 6.1. Our model characterizes the service level provided by N independent duplicates running the exact same workload. The latency bound (τ) for the SLO is given as input. Written in plain English, *our model predicts that $\hat{s}$ percent of requests will complete within τ ms.*

$\hat{s}$	Expected service level
N	Number of duplicates used to mask anomalies
τ	Target latency bound
$\Phi_n(k)$	Percentage of service times from duplicate n with latency below k
λ	Mean interarrival rate for storage accesses
μ_{net}	Mean of network latency between duplicates and storage clients
μ_{rep}	Mean delay to duplicate a message one time plus the delay to prune a tardy reply
μ_n	Mean service time for duplicate n (derived)

Table 6.1: Zoolander inputs.

Using principles #1 and 2, we first model the probability that the fastest duplicate will meet an SLO latency bound. Recall, writes are sent to all duplicates, so any duplicate can process any request. Handling failures is treated as an implementation issue, not a modeling issue. The probability that the fastest duplicate responds within latency bound is computed as follows:

$$\hat{s} = \sum_{n=0}^{N-1} [\Phi_n(\tau) * \prod_{i=0}^{n-1} (1 - \Phi_i(\tau))]$$

To provide intuition into this result, consider $\Phi_i(\tau)$ is the probability that duplicate i meets the τ ms latency bound. If $N = 2$, $\Phi_1(\tau) * (1 - \Phi_0(\tau))$ is the probability that duplicate 1 masks a SLO violation for duplicate 0. Intuitively, as we scale out in N, each term in the sum is the probability that the n^{th} duplicate is the firewall for meeting SLO, i.e., duplicates $0..(n-1)$ take too long to respond but n meets the bound. When all duplicates have the same service time distribution, we can reduce the above equation to a geometric series, shown below. (Note, as N approaches infinity, $\hat{s}$ converges to 1.)

$$\hat{s} = \sum_{n=0} \Phi_n(\tau) * (1 - \Phi_n(\tau))^n = 1 - (1 - \Phi)^N$$

Queuing and Network Delay: SLOs reflect a client's perceived latency which may include processing time, queuing delay and network latency. Since duplicates execute the same workload, they share access arrival patterns and their respective queuing delays are correlated. Similarly, networking problems can affect all duplicates. Here, we lean on prior work on queuing theory to answer two questions. First, does the expected queuing level completely inhibit replication for predictability? And second, how many duplicates are needed to overcome the effects of queuing? The key idea is to deduct the queuing delay from τ in the base model. Intuitively, requiring all duplicates to reduce their expected service time in proportion to the expected queuing delay.

$$\tau_n = \tau - (\frac{1 + C_v^2}{2} * \frac{\rho}{1 - \rho} * \mu_n) - \mu_{net}$$

$$\hat{s} = \sum_{n=0}^{N-1} [\Phi_n(\tau_n) * \prod_{i=0}^{n-1} (1 - \Phi_i(\tau_i))]$$

We used an M/G/1 queuing model to derive the expected queuing delay, reflecting the heavy-tail service times observed in Figure 6.2a. To briefly explain the first equation above, an

M/G/1 models the expected queuing delay as a function of system utilization (ρ), distribution variance (C_v^2), and mean service time. Utilization is the mean arrival rate divided by the mean service time. Note, that the new τ may be different for each node (parameterizing it by n). An M/G/1 assumes that inter-arrivals are exponentially distributed. This may not be the case in all data-intensive services. A G/G/1 with some constraints on inter-arrival may be more accurate. Alternatively, an M/M/1 would have simplified our model, eliminating the need for the squared coefficient of variance (C_v^2). Prior work has shown that multi-class M/M/1 can sometimes capture the first-order effects of M/G/1.

We deduct the mean time lost to network latency. Here, network latency is the average delay in sending a TCP message between any two nodes.

Multicast and Pruning Overhead: Replication for predictability incurs overhead when messages are repeated to all duplicates and when unused messages are pruned. These activities become more costly as the number of duplicates increase. We use a linear model to capture this. Note, we expect emerging routers to provide multicast support that reduces this overhead a lot. However, storage systems that use software multicast, like Zoolander should consider this overhead.

$$\tau_n = \tau - (\frac{1+C_v^2}{2} * \frac{\rho}{1-\rho} * \mu_n) - \mu_{net} - N * \mu_{rep}$$

Discussion: With a nod toward systems builders, we kept the model simple and easy to understand. Most inputs come from CDF or arrival-rate data that can be collected using standard tools. The model does not capture non-linear correlations between outliers, resource dependencies, or the root causes of SLO violations.

6.4 Zoolander

Zoolander is middleware for existing key-value stores. It adds full read and write support for replication for predictability. Figure 6.3 highlights the key components of Zoolander. In the center of the figure, we show that keys are stored in *duplicates and partitions*. A duplicate abstracts an existing key-value store, e.g., Zookeeper or Cassandra. As such, a duplicate may span many nodes but it does not share resources with other duplicates.

A partition comprises 1 or more duplicates. Storage accesses are sent to all duplicates within a partition—i.e., duplicates implement replication for predictability. Storage accesses are sent to only 1 partition. There is no cross-partition communication. A global hash function maps keys to partitions. All of the keys mapped to a partition comprise a *shard*.

Zoolander can scale out by reducing storage accesses per node via partitioning. It can also scale out by adding duplicates. At the top of Figure 6.3, we highlight the Zoolander manager which uses our analytic model to scale out efficiently. The manager takes as input a target service level and latency bound. It also collects CDF data on service times, networking delays, and arrival rates per shard. The manager then uses our model from Section 6.3 to find a replication policy that meets the target SLO. It finds a policy by iteratively 1) moving a shard from one partition to another, 2) placing a shard on a new partition, and 3) adding/removing duplicates from a partition. The first and second options change the arrival rate for each partition and are captured by our queuing model. The third option is captured by our geometric series.

6.4.1 Consistency Issues

A read after write to the same key in Zoolander returns either a value that is at least as up to date as most recent write by the client (read my own write) or the value of an

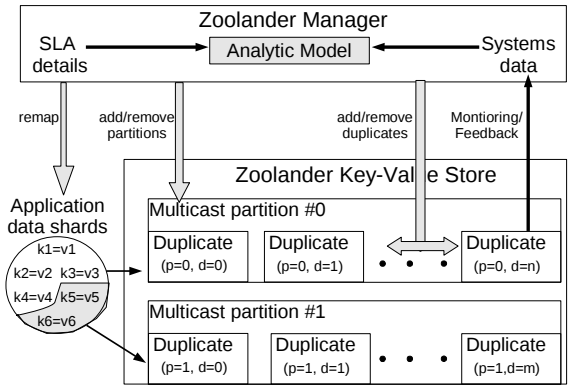


Figure 6.3: The Zoolander key value store. SLA details include a service level and target latency bound. Systems data samples the rate at which requests arrive for each partition, CPU usage at each node, and network delays. We use the term *replication policy* as a catch all term for shard mapping, number of partitions, and the number of duplicates in each partition.

earlier, valid write (eventual). We can also support strong consistency funneling all accesses through a single multicast node. However, we rarely use strong consistency in any Zoolander deployments. As many prior works have noted [50, 76, 101, 152], read-my-own writes and eventual consistency normally suffice.

To support read-my-own-write consistency, each duplicate processes puts in FIFO order. Gets (reads) may be processed out of order. Clients accept reads only if the version number exceeds the version produced by their last write. For eventual consistency, Zoolander clients ignore version numbers. Figure 6.4 clearly depicts the supported consistency. Read my own write avoids stale data but gives up redundancy.

Propagating Writes: To ensure correct results, writes must propagate to every duplicate and every duplicate must see writes in the same order. Zoolander achieves this by using multicast. Zoolander's *client side* library keeps IP addresses for the head node of each duplicate. When client's issue a put request, the library issues D identical messages to

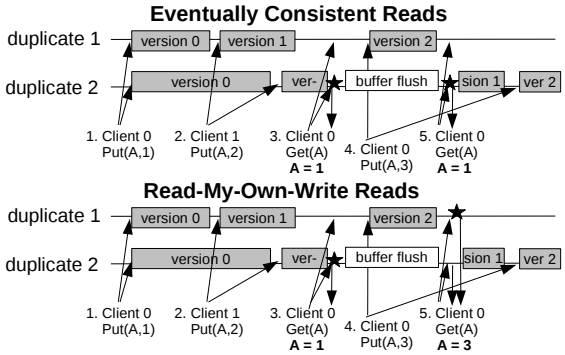


Figure 6.4: Version based support for read-my-own-write and eventual consistency in Zoolander. Clients funnel puts through a common multicast library to ensure write order. The star shows which duplicate satisfies a get. Gets can bypass puts.

each duplicate in a globally fixed order. In the future, we hope to replace this library with networking devices with hardware support for multicast.

Software multicast ensures that writes from a single client arrive in order, but writes from different clients can arrive out of order. We assume that multiple clients racing to update the same key is *not* the common case. As such, Zoolander provides a simple but costly solution. To share keys, clients funnel writes through a master multicast client. This approach sacrifices throughput but ensures correct results (see Figure 6.4).

Choosing the Right Store: By extending existing key value stores, Zoolander inherits prior work on achieving high availability and throughput. The downside is that there are many key value stores; each tailored for high throughput under a certain workload. Zoolander leaves this choice to the storage manager. In our tests, the default store is Zookeeper [76] because of its wait-free features. However, for online services that need high throughput and rich data models [43, 44, 63], we extend Cassandra [74]. We have also run tests with in-memory stores Redis and Memcached.

Relative Throughput & Processing Overhead		
Writes	Read-my-own-write Reads	Eventual Reads
95%(48us)	94%(52us)	99%(<1us)

Table 6.2: Zoolander's maximum throughput at different consistency levels relative to Zookeeper's [76]. In parenthesis, average latency for multicast and callback.

6.5 Implementation

Overhead: Our software multicast is on the data path of every write; It must be fast. Our multicast library avoids TCP handshakes by maintaining long-standing TCP connections between clients and duplicates. Also, Zoolander eschews costly RPC in favor of callbacks. Clients append a writeback port and IP to every access that goes through our multicast library. Duplicates respond to clients directly, bypassing multicast. We measured the maximum number of writes, read-my-own reads, and eventual reads supported per second in Zoolander with Zookeeper as the underlying store. Table 6.2 compares the results to the throughput of Zookeeper by itself [76]. These tests were conducted on our private cloud.

Bandwidth: Each duplicate receives the same workload and uses the same network bandwidth. At scale, duplicates could congest data center networks. Zoolander takes 2 steps to use less bandwidth. First, writes return only "OK" or "FAIL", not a copy of data. Second, for reads, Zoolander re-purposes replicas set up for fault tolerance as duplicates. Such replicas are common in production [50, 152]. Figure 6.5(A) compares the bandwidth used by naive support for replication for predictability against Zoolander's approach. The baseline is the bandwidth consumed by a 3-node quorum system [50, 152]. Our approach lowers bandwidth usage by 2X.

Dynamic Systems Data: Zoolander continuously collects data using sliding windows. To keep overhead low, we collect data for only a random sample of storage accesses. For each sampled access, we collect response time, service time, accessed shard number, and network latency. A window is a fixed number of samples.

We compute the mean network latency and arrival rate for each window. We use the information gain metric to determine if our CDF data has diverged. If we detect that the CDF may have diverged, we collect samples more frequently, waiting for the information gain metric to converge on new CDF data. Figure 6.5 demonstrates the benefits of service time windows. First, we ran our e-science workload (Gridlab-D), then we injected an additional write-only workload on the same machine, and finally, we added a read-only workload also. Our sliding windows allow us to capture accurate service time distributions shortly after each injection, as shown by convergence on information gain.

Fault Tolerance: Zoolander can tolerate duplicate, partition, software multicast, and client failures. Duplicate failures are detected via TCP Keep Alive by the software multicast. Every duplicate receives every write, so between storage accesses, software multicast can simply remove any failed duplicate from the multicast list.

A partition fails when its only working duplicates fails. When this happens, Zoolander manager uses transaction logs from the last surviving duplicate to restart the partition. This takes minutes but is automated. Software multicast is a process in the client-side library. On restart, it updates its multicast list with Zoolander manager. This process takes only milliseconds. However, when software multicast is down, the entire partition is unavailable.

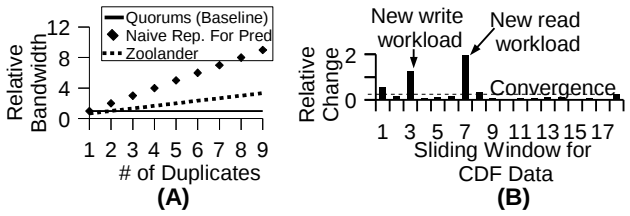
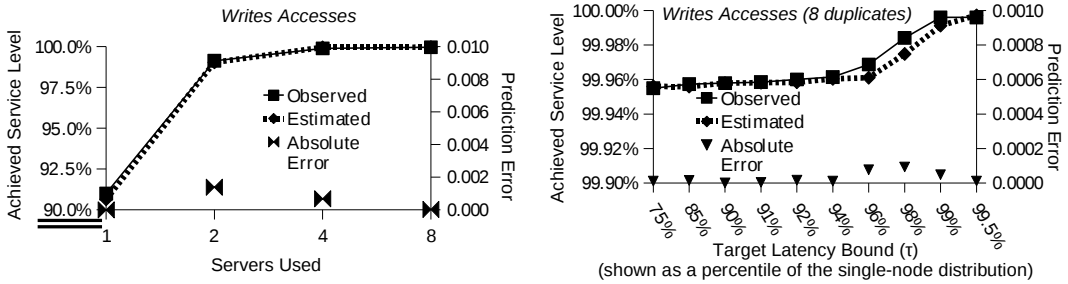
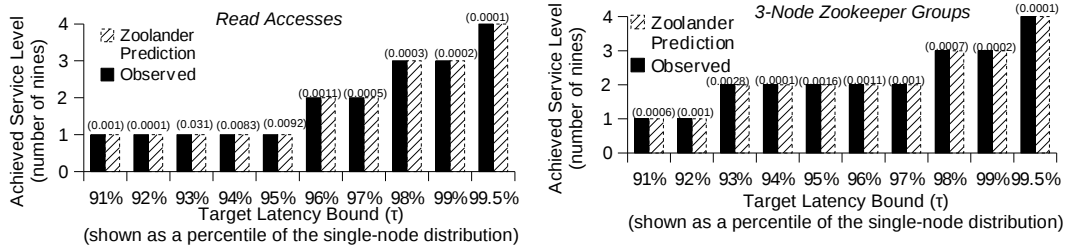


Figure 6.5: (A) Zoolander lowers bandwidth needs by re-purposing replicas used for fault tolerance. (B) Zoolander tracks changes in the service time CDF relative to internal systems data. Relative change is measured using information gain.



(a) Achieved service levels against Zoolander predictions as duplicates increase. Observed and estimated lines overlap.

(b) Service levels as the target latency bound changes.



(c) Service levels achieved on read-only accesses. 2 duplicates used.

(d) Service levels achieved under 3-node Zookeeper deployment. 2 duplicates used.

Figure 6.6: Validation Experiments

6.5.1 Model Validation & System Results

Thus far, we have developed an analytic model for replication for predictability. We have also described the system design for Zoolander, a key value store that fully supports replication for predictability at scale. Here, we show that Zoolander achieves performance expected by our model and that the model has low prediction error.

We deployed Zoolander on the private cloud described in Section 6.3. We used Zookeeper as the underlying key-value store. We focus on data sets that fit within memory (i.e., in-memory key-value stores backed up with local disk). We used 1 partition for these tests. We issued 1M write accesses in sequence without any concurrency. We used the 90^{th} percentile of the collected service time distribution as the default latency bound (τ=5ms). The average response time in this setup was 3ms, so our latency bound allowed only 2ms for outliers. The SLO for Zookeeper without Zoolander was: *90% of accesses will complete within 5ms*.

We added duplicates to Zoolander one at a time, issuing the same write workload each time we scaled out. Figure 6.6a shows Zoolander's performance, i.e., achieved service level, as duplicates increase. Specifically, the achieved service level grew as duplicates were added. For example, under 8 instances, Zoolander could support the following SLO: *99.96% of write accesses will complete within 5ms*. The graph also shows that Zoolander had absolute error (i.e., actual service level minus predicted) below 0.002 in all cases. **This is a key result: Scaling out via replication for predictability strengthens SLOs without raising latency bounds.**

In our next test, we set the number of duplicates to 8. We used the same service time distribution from above. We then changed the latency bound (τ) to different percentiles in the single-node distribution, from the 75^{th} to 99.5^{th}. High percentiles led to several-nine service levels in Zoolander, forcing our model to be accurate with high precision. Low

percentiles required Zoolander to accurately model more accesses. Figure 6.6b shows our model's accuracy as the latency bound increased. The absolute error was within 0.0001 for high and low percentiles. In Figure 6.2a, we observed that write access times had a heavy-tail distribution that started around the 96^{th} percentile. Figure 6.6b shows a steeper slope (strong gains) for latency bounds after the 96^{th} percentile. For instance, setting the latency bound to the 99^{th} percentile of single-node distribution (τ=15ms), RP Zookeeper achieved 99.991% service level using only 4 duplicates. In other words, adding duplicates scaled the service level by two orders of magnitude.

6.6 Model-Driven SLO Analysis

Zoolander can scale out via replication for predictability or via partitioning. The analytic model presented in Section 6.3 helps Zoolander manager choose the most efficient replication policy. The analytic model can also provide marginal analysis on the SLO achieved as key input parameters vary. Specifically, we varied the request arrival rate and used our model to predict SLO achieved. We fixed the number of nodes (4) and we fixed the systems data. We compared 3 replication policies: 1) using only replication for predictability (i.e., 1 partition with 4 duplicates), 2) using only traditional approaches (i.e., 4 partitions with 1 duplicate each), and 3) using a mixed approach (i.e., 2 partitions and 2 duplicates each). Note, our model predicts the same service levels under a k-duplicate partition with arrival rate λ as it does under N k-duplicate partitions with an arrival rate $N * \lambda$, making our results relevant to larger systems.

Recall, our model is biased toward partitioning. We naively assume that each partition divides workload evenly with no internal hot spots or convoy effects. Thus, we are really comparing accurate predictions on replication for predictability to best-case predictions for

partitioning. More generally, our model makes best-case predictions for any approach that reduces accesses per node by dividing work, including replication for throughput.

The results of our marginal analysis are shown in Figures 6.7(a–b). The y-axis in these figures is "goodput", i.e., the fraction of requests returned within SLO. The x-axis for these figures is the normalized arrival rate, i.e., the arrival rate over the maximum service rate. In queuing theory terminology, the normalized arrival rate is called system utilization. The latency bound changes across the figures. The results show arrival rates under which the studied replication heuristics excel. Specifically:

1. Zoolander's mixed approach, using both replication for predictability and partitioning offers the best of both worlds. Replication for predictability alone increased service levels but only under low arrival rates. Partitioning alone supported high arrival rates but with low service levels. The mixed approach supported high arrival rates ($>$40% utilization) and achieved high SLO.

2. As the latency bound increased, replication for predictability supported higher arrival rates, and similarly, partitioning provided higher service levels.

3. Replication for predictability performs horribly under high arrival rates. Recall, all duplicates have the same queuing delay, once this delay exceeds the latency bound, replication for predictability offers no benefit. It's performance falls off a cliff.

4. Divide-the-work approaches simply can't achieve high service levels under tight latency bounds. When we set $\tau = 3.5$ms, goodput under traditional only fell below 94%. A mixed approach achieved 99% goodput.

Cost Effectiveness: SLO violations can be costly. For online e-commerce services, violations reduce sales and ad clicks. For data processing services, violations deprive business

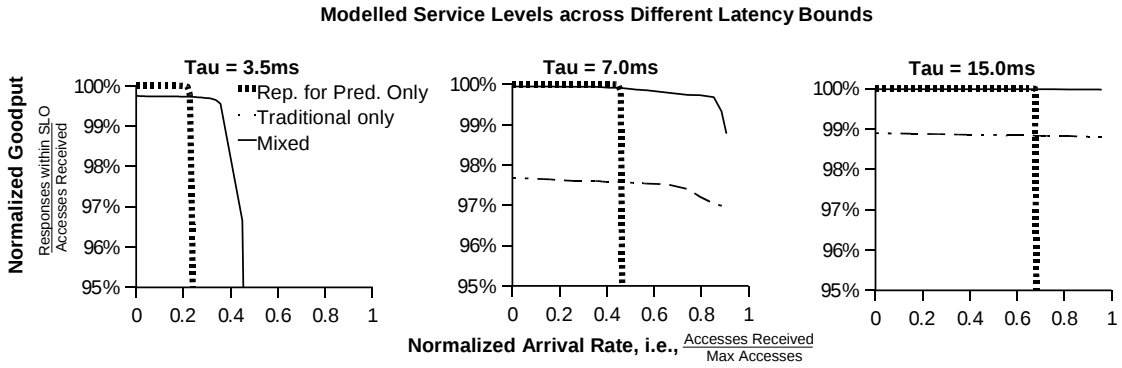


Figure 6.7: Trading throughput for predictability. For replication for predictability only, $\lambda = x$ and $N = 4$. For traditional, $\lambda = \frac{x}{4}$ and $N = 1$. For the mixed Zoolander approach, $\lambda = \frac{x}{2}$ and $N = 2$. Our model produced the Y-axis.

leaders of data needed to make good, profitable choices. All else being equal, reducing SLO violations means reducing costs. Replication for predictability reduces SLO violations but it uses more nodes. Nodes also cost; They use energy, their components (memory and disk) wear out, and they have management overheads. We used our model to study the cost effectiveness of using more nodes to reduce SLO violations.

We set a latency bound (τ) of 7ms and used systems data taken from our private cloud. We computed the number of SLO violations as the arrival rate changed. To provide intuition, the number of SLO violations is essentially the product of x and $(1-y)$ for (x,y) pairs in Figure 6.7b. The rate of violations (λ^{vio}) is shown below. F_{zk} represents our model with systems data from Zookeeper.

$$\lambda^{vio} = \lambda * (1 - F_{zk}(\tau, \lambda))$$

We used a linear model to assess cost effectiveness. Total cost was the sum of 1) SLO violations (λ^{vio}) multiplied by cost per violation (cpv) and 2) nodes used (N) multiplied by

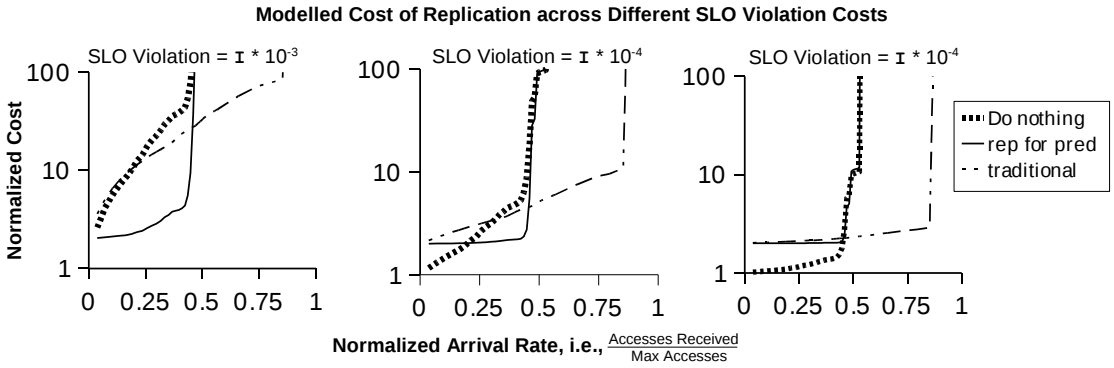


Figure 6.8: Cost of a 1-node system, 2 partition system, and 2 duplicate system across arrival rates. Lower numbers are better.

the cost per node per unit time (cpn). The model is shown below.

$$cost = N * cpn + \lambda^{vio} * cpv$$

The cost per violation and cost per node vary from service to service. We studied the relative cost between these parameters. Specifically, we set $cpn = 1$ and varied cpv, as shown in Figures 6.8(a-b).

We compared three replication policies. The default approach, or "do nothing", did not scale out. It used 1 1-duplicate partition ($N = 1$) and allowed SLO violations to increase with the arrival rate. The replication for predictability approach used 1 2-duplicate partition ($N = 2$) and reduced SLO violations under low arrival rates. The traditional approach used 2 1-duplicate partitions ($N = 2$). Note, N refers to the number of duplicates—each duplicate could comprise many nodes. We found the following insights:

1. When 100 SLO violations cost more than a node, replication for predictability is cost effective until queuing delay exceeds the latency bound and service levels fall of the cliff.

2. If SLO violations are cheap, e.g., a node costs more than 10,000 violations, replication for predictability is never cost effective, even under low arrival rates.

3. If arrival rates change, the most cost effective approach will also change. When SLO violations are neither cheap nor expensive, all three approaches can be cost effective at certain rates.

The exact cost of an SLO violation depends on the service. Online services have found ways to compute *cpv* for their workloads. It is harder to compute *cpv* in emerging services, e.g., Twitter trend analysis or smart-grid power management. In these services, violations map only indirectly to revenue. However, if such violations lead to stale results that lead to poor decisions, the real cost of such violations can be very high.

6.7 Evaluation

For this section, we studied Zoolander under intense arrival rates, e.g., workloads produced by online services. These tests used up to 144 Amazon EC2 units. EC2 is widely used by e-commerce sites and web portals. It's prices are well known. Our goal was to compare Zoolander scaling strategies and to highlight real world settings where replication for predictability is cost effective.

Many online services see diurnal patterns in the arrival rates of user requests [13, 142]. Request arrival rates can fall by 50% between 12am–4am compared to daily peaks between 9am–7pm. As a result, fewer nodes are needed in the night than in the daytime. Nonetheless, services must buy enough nodes to provide low response times under peak arrival rates. Some services save energy by using only a fraction of their nodes during the night, turning off unused nodes. However, in data centers, energy costs are low at night (because demand for electricity is low). Nighttime energy prices below $0.03 are common. The typical service

would save only $0.12 per night by turning off a 1KW server during this period. Zoolander can exploit underused nodes in a different way; Turning them into duplicates to reduce SLO violations.

We compared the opportunity costs of reducing SLO violations against turning off machines. Figure 6.9 shows the competing replication policies. During the daytime, each node is needed for high throughput and operates under its max arrival rate. However, at nighttime, the arrival rate drops by 50%, allowing us to place 2 shards on 1 node or to use replication for predictability. To save energy at night, our replication policy consolidates shards, using as few nodes as possible without exceeding the peak per-node arrival rate. Our workload accesses all shards evenly, i.e., no hot spots.

Replication for predictability can be applied naively on top the energy saving approach by using idle nodes as duplicates. SCADS manager adopted this approach [152]. However, our findings in Section 6.6 suggest that arrival rates on nodes that use replication for predictability should be low. We decided to use replication for predictability more sparingly, keeping arrival rates low for the duplicates. For every 6 nodes, we placed 4 shards on 2 nodes (like in the energy saving approach). The remaining 4 nodes hosted 2 shards via 2 2-duplicate partitions. Our approach had 9.7% fewer violations compared to the naive approach described above.

To make the test realistic, we setup Zoolander on EC2 and tried to mimic the scale of TripAdvisor's workload. Public data [63] shows that TripAdvisor receives 200M user requests per day. On average, each user request accesses the back-end Memcached store 7 times, translating to 1.4B storage accesses per day. Learning from recent studies, we assumed the arrival rate would drop by 50% [13]. Our goal was to support 29M accesses per hour.

We used 48 clients that issued a mix of 15% Gets and 85% Puts across 96 shards. Gets/Puts were issued in batches of 20, reflecting correlated storage accesses within user requests. Each batch arrived independently, leading to exponentially distributed inter-arrival times. Note, our clients followed a realistic, open-loop workload model. Duplicates in these tests were 1-node Cassandra [74]. In a 4 hour test, our clients issued over 160M key-value lookups (40M per hour).

During our tests, Zoolander achieved high throughput and fault tolerance. While these metrics do not reflect Zoolander's contribution, they are not weak spots either! To support 40M lookups per hour, Zoolander used 48 EC2 compute units with Cassandra as the underlying key-value store.

Peak throughput was 431 lookups per second per EC2 unit, about 20% higher than the average achieved by Netflix operators [43]. We encountered 546 whole partition failures across the 144 nodes where either Cassandra or the software multicast crashed. During those failures, 2,929 lookups failed. Multicast, duplicates or callbacks caused 1,200 of those failed lookups.

SLO violations also occur when Zoolander migrated data to its nighttime setup. Migrations periods are shown in Figure 6.10(a). The figure is based on a trace from [142]. Moving to from the daytime setup to Zoolander's nighttime setup needed 4 shard migrations(see Figure 6.9). Moving back to daytime setup needed 2 more migrations. Zoolander used existing techniques for shard migration. We measured migration-induced violations under load and added these to Zoolander's costs.

We used the cost model in Section 6.6 to compare the nighttime replication policies in Figure 6.9. We studied two different cost per node settings. In the *private cloud* setting, the cost of a node is a function of its energy usage only. We assumed a cost of $0.03KWh and

node id	daytime setup	night time energy saver	night time Zoolander
0	accesses/hr = 1.6M hosted shard(s) = A	accesses/hr = 1.6M hosted shard(s) = A,B	accesses/hr = 1.6M hosted shard(s) = A,B
1	accesses/hr = 1.6M hosted shard(s) = B	accesses/hr = 1.6M hosted shard(s) = C,D	accesses/hr = 1.6M hosted shard(s) = C,D
2	accesses/hr = 1.6M hosted shard(s) = C	accesses/hr = 1.6M hosted shard(s) = E,F	accesses/hr = 0.8M hosted shard(s) = E
3	accesses/hr = 1.6M hosted shard(s) = D	NOT USED	accesses/hr = 0.8M hosted shard(s) = E
4	accesses/hr = 1.6M hosted shard(s) = E	NOT USED	accesses/hr = 0.8M hosted shard(s) = F
5	accesses/hr = 1.6M hosted shard(s) = F	NOT USED	accesses/hr = 0.8M hosted shard(s) = F

Figure 6.9: Replication strategies during the nighttime workload for an e-commerce service.

that each node (an EC2 unit) used 100W, thus $cpn = \$0.003$. In the public cloud setting, the cost of a node includes everything provided by EC2 (i.e., high availability, EBS, etc). As of this writing, cpn of a small EC2 unit was \$0.085 (20X more than energy only costs). The energy saving approach used 3 nodes, whereas the Zoolander approach used 6. We set the SLO latency bound (τ) for a batch of lookups to 150ms. Out of 160M requests the Zoolander approach incurred only 57K SLO violations compared to 85K in the energy saving approach—a reduction of 32%.

Figure 6.10(b) plots relative cost as a function of cost per 1000 violations (cpv). Lower numbers are better for Zoolander. The x-axis is log scale base 10. As SLO costs increase, the Zoolander approach becomes more cost effective. Without considering migration costs, the relative cost converges to 68% quickly in the private cloud setting where cpn is very low. Migration costs increase the relative cost by 16%, but Zoolander remains highly cost effective in private clouds. Figure 6.10(b) shows that Zoolander spends \$0.79 to every dollar spent by energy saver approach, saving 21%. The public cloud setting requires higher cpv to be cost effective.

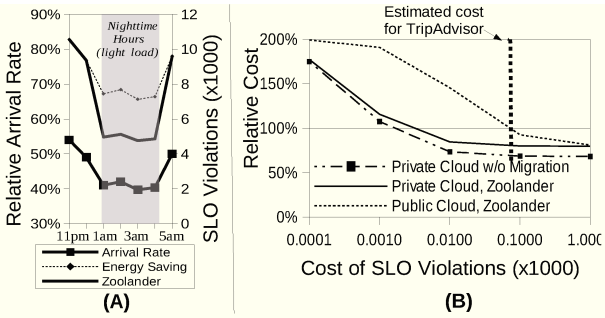


Figure 6.10: (A) Zoolander reduced violations at night. From 12am-1am and 4am-5am, Zoolander migrated data. We measured SLO violations incurred during migration. (B) Zoolander's approach was cost effective for private and public clouds. Relative cost is $(\frac{\text{zoolander}}{\text{energy saver}})$

In their fiscal statement for the 4^{th} quarter of 2011, TripAdvisor earned \$122M from click- and display-based advertising. We divided this number by 200M daily user requests to get revenue per 1,000 page views of \$6.81. Using prior research, we estimated that each SLO violation (a 100ms delay) would lead to a 1% loss in profits [129], meaning $cpv = \$0.068$.

Under this setting, the Zoolander approach was cost effective for private settings and broke even with the energy savings approach under public cloud settings. When we consider migration costs for the energy savings approach, Zoolander is cost effective even for public clouds.

Model-Driven Management The nighttime policy for the EC2 tests was a heuristic based on insights from Section 6.6. Heuristics derived from principled models underlie many real world systems. Alternatively, Zoolander's model can be queried directly to find good policies.

We used systems data from Zookeeper and set τ to 3.5ms, a very low latency bound. We studied the hourly arrival rates (λ) shown in Table 6.3. For each rate, our model computed the expected SLO under 8 policies: 8 partitions(p) each with 1 duplicate(d), 4p with 2d, 2p

with 4d, 6p with 1d, 3p with 2d, 2p with 3d, 4p with 1d, and 2p with 2d. Table 6.3 shows the policy that met SLO using the fewest nodes. The 5 policies shown all differ, including policies with more than 2 duplicates.

Target SLO:	*98% of accesses complete in 3.5ms*				
Accesses/Hour:	2K	850K	1M	1.5M	1.9M
Best Policy:	4p/1d	2p/2d	2p/3d	3p/2d	4p/2d

Table 6.3: Best replication policy by arrival rate

6.8 Summary

This chapter presented Zoolander, middleware that fully supports replication for predictability on existing key value stores. Replication for predictability redundantly sends each storage access to multiple nodes. By doing so, it sacrifices throughput to make response times more predictable. Our analytic model explained the conditions where replication for predictability outperforms traditional, divide-the-work approaches. It also provided accurate predictions that could be queried to find good replication policies. We tested Zoolander with Zookeeper and Cassandra. Its overhead was low. Our largest test (spanning 144 EC2 compute units) showed that Zoolander achieved high throughput and strengthened SLOs. By wisely mixing scale-out approaches, Zoolander reduced operating costs by 21%.

Chapter 7: Conclusions

In this chapter, we conclude by restating the problem we tried to address and summarize our key contributions pertaining to the problem statement. Finally, we describe some future directions that can overcome the limitations of the current thesis.

With the exponential increase of data collected from different domains, there is a need for novel ideas of scaling algorithms to those large datasets. The underlying reason for this requirement is that traditional scaling techniques such as increase of processor clock speed have halted due to the end of Denard Scaling. In this thesis, we investigate the opportunities of scaling analytics to large datasets using approximate and distributed computing techniques. Prior works have studied both approximate and distributed computing techniques, however, the following characteristics make our research novel.

- Most approximate computing techniques come with guarantees (e.g. PCA guarantees data reconstruction error). However, we investigated the challenge of providing theoretical guarantees in terms of application specific quality metrics and generalizing them.

- Distributed computing models (e.g. MapReduce) are usually studied as general purpose solutions. We investigated distributed computing models specifically for analytics and showed that the payload characteristics have a significant impact on

the performance of distributed analytics tasks. We investigated the opportunities of guiding the partitioning of data by understanding the data characteristics using methods from the approximate computing domain.

7.1 Summary of Key Contributions

7.1.1 LSH for APSS

We first solve the APSS task using approximate computing. We use the popular hashing based dimensionality reduction technique called LSH. The theoretical guarantee resulting from LSH is that the hash collision probability is equal to the similarity of a pair of data points. Consequently, similar points have similar hash sketches. However, the quality metrics of APSS is typically recall or precision. We developed a principled technique for the APSS task such that the algorithm will guarantee the required recall and precision while automatically tuning the number of hash functions to achieve that accuracy.

7.1.2 LSH for Kernels

In many complex application domains such as image processing, text analytics, and bioinformatics, vanilla similarity measures (for which unbiased LSH estimators exist) such as Euclidean distance, Jaccard index, and cosine similarity do not work well. Those domains require specialized similarity measures under which data points are better separable. Kernel similarity measures are a class of inner product similarity that is extremely powerful and flexible in explaining the structure of data. We developed a novel data embedding that leads an unbiased LSH estimator for arbitrary kernel similarity measures. We believe this generalization to arbitrary kernels is key to applying LSH based approximate solutions to different complex domains.

7.1.3 Distributed Anomaly Detection

Here we show that approximating the input data (graph) can play a key role in controlling the algorithmic complexity of downstream inference task. Coupled with distributed execution, this helps us scale the anomaly detection task to a sensor network graph consisting of hundreds of millions of edges. We represent the sensor network as an MRF and use LBP for the inference. We show that an approximation in the construction of the sensor network graph can significantly improve the convergence of vanilla LBP.

7.1.4 Distributed Framework for Analytics

Distributed computing frameworks suffer performance degradation due to load imbalance. This load imbalance is a consequence of heterogeneity of the data center environment. Data centers are increasingly becoming heterogeneous in terms of processing capacity and renewable energy availability due to a number of reasons such as server upgrades, power constrained operation and virtualization. However, a key insight we investigate is that the payload characteristics of the individual partitions can cause significant load imbalance as well. We develop a technique to control the data partitions such that the partitions will have similar statistical characteristics. For a large class of analytics tasks, this results in load balanced operations. Key to our technique is the approximate characteristics analysis of the entire data through LSH. We build a general purpose distributed analytics framework sitting on top of NoSQL stores that can partition and place data while taking into account both the environment heterogeneity as well as the data characteristics.

7.1.5 Distributed NoSQL stores

Finally, we investigate the choice of distributed NoSQL stores as the underlying storage for analytics workload requiring a realtime response. The key insight we found is that all NoSQL stores exhibit the heavy-tailed latency distribution. Therefore, even though the average response times are extremely fast, there are some requests which are significantly slower. Such requests are prohibitive for interactive workloads. We design a middleware that can sit on top of any NoSQL store and combines replication for predictability and replication for throughput in a principled way using probabilistic arguments and queueing theory. Our framework can guarantee tail latencies (99^{th} percentile) and not just average latencies.

To conclude, our key insight is that even though approximate computing techniques can scale analytics tasks to large datasets, such scaling up is only meaningful when simultaneously rigorous quality guarantees in terms of application specific quality metrics can be given. Additionally, generalizing such tasks to different domains poses several challenges. Finally, for analytics tasks techniques for approximate computing methods can provide insights on efficient execution of distributed analytics.

7.2 Future Works

The key insights derived from the research in this thesis opens up interesting research directions both in the approximate computing and the distributed computing space for analytics tasks.

7.2.1 Approximate Computing

Our key insight in this space is in spite of the existence of a number of approximate data size reduction techniques with some form of theoretical guarantee (e.g. data reconstruction

error for PCA), their usage is limited to specific applications as it is unclear how those techniques impact the application specific quality metrics. We believe to use these approximation techniques effectively, there is a need to guarantee the application specific metrics (rather than the quantity that is optimized by the approximation technique). This needs to be done in a case by case basis and as such provides a lot of future work opportunities.

We would like to extend our techniques to a closely related (to APSS) problem, the top-k nearest neighbor problem [8]. LSH already supports such queries. We believe our pruning technique can be applied to further improve the performance of LSH based techniques. Another aspect to investigate is the choice of the hash function in the context of dimensionality reduction. LSH has the nice property of being data independent and its probabilistic guarantees help us design the algorithm to guarantee recall for APSS. However, if the input data is not uniformly distributed, LSH starts suffering from performance bottlenecks. In fact, if the input data distribution is extremely skewed, then LSH may perform as bad as exhaustive search. A recent line of work, data dependent hashing [10, 11] learns the hash function from the distribution of data and consequently tries to distribute input data uniformly across hash bucket. However, many of the nice LSH properties become difficult to achieve, making it harder to relate these approximate techniques to application specific quality metrics. We believe this is an important future research direction. Furthermore, another important direction is the generality of the developed methods. Since there are many analytics tasks, providing guarantees in an application specific manner is not feasible. We need to identify classes of algorithms that can share similar methods to provide rigorous guarantees. For instance, APSS and top-k both require recall/precision guarantees and we believe this can be achieved by a unified framework.

7.2.2 Distributed Computing

We developed a general purpose framework that can partition and place data for distributed analytics in a performance, energy, and payload aware way. We showed the performance benefits for frequent pattern mining and webgraph compression. Here an interesting thing to study will be to characterize the analytics tasks that can benefit from out partitioning strategies. Additionally, we believe that our framework may have qualitative benefits along with performance benefits. For instance, the ApproxHadoop [69] framework partitions the data using cluster sampling and shows good performance and quality for aggregation workloads. Our partitioning strategy relies on stratified sampling instead. It would be interesting to compare the quality of stratified sampling against ApproxHadoop's cluster sampling for aggregation workloads. Finally, there is a scope of improvement in the modeling step. We use a simple linear model for the Pareto modeling step. However, in many workloads, this linear assumption may not be satisfied. The problem with using non-linear models is that (i) they tend to overfit and (ii) they need a large number of samples to fit. Collecting samples is an expensive operation as it requires us to run the target workload on a sample of the input data. Improving the model will be an important direction for future research. Furthermore, our method can be thought of as a preprocessing step that tells us how to partition the data for effective load balanced execution. However, after the workload starts executing, the processing parameters may change to a number of reasons (e.g. colocation of multiple workloads, sudden unavailability of renewable energy etc.). It may be very useful to combine dynamic load balancing techniques with our methods so that our framework can rapidly respond to such changes. The challenge in this space is to do it in a data-aware way. This is paramount to analytics tasks.

Bibliography

[1] Law lab datasets. http://law.di.unimi.it/datasets.php/.

[2] Pvwatts simulator. http://pvwatts.nrel.gov/.

[3] Uw xml repository. http://www.cs.washington.edu/research/xmldatasets/.

[4] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[5] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery*, 29(3):626–688, 2015.

[6] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: Attack of the clones. In *USENIX NSDI*, 2013.

[7] Ganesh Ananthanarayanan, Srikanth Kandula, Albert Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. Reining in the outliers in map-reduce clusters using mantri. In *USENIX OSDI*, 2010.

[8] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[9] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51:117–122, 2008.

[10] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 793–801. ACM, 2015.

[11] Alexandr Andoni and Ilya Razenshteyn. Tight lower bounds for data-dependent locality-sensitive hashing. *arXiv preprint arXiv:1507.04299*, 2015.

[12] M. Armbrust, A. Fox, D. Patterson, N. Lanham, H. Oh, B. Trushkowsky, and J. Trutna. Scads: Scale-independent storage for social computing applications. 2009.

[13] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload analysis of a large-scale key-value store. In *ACM SIGMETRICS*, 2012.

[14] M. Attariyan, M. Chow, and J. Flinn. X-ray: Automating root-cause diagnosis of performance anomalies in production software. In *USENIX OSDI*, 2012.

[15] P. Bailis. Doing redundant work to speed up distributed queries. `http://www.bailis.org/blog/`.

[16] Bharathan Balaji, Chetan Verma, Balakrishnan Narayanaswamy, and Yuvraj Agarwal. Zodiac: Organizing large deployment of sensors to create reusable applications for buildings. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, BuildSys '15, pages 13–22, New York, NY, USA, 2015. ACM.

[17] Bortik Bandyopadhyay, David Fuhry, Aniket Chakrabarti, and Srinivasan Parthasarathy. Topological graph sketching for incremental and scalable analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1231–1240. ACM, 2016.

[18] R.J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, 2007.

[19] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, Albert Zomaya, et al. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers*, 82(2):47–111, 2011.

[20] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[21] Robert D Blumofe and Charles E Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM (JACM)*, 46(5):720–748, 1999.

[22] Tom Bohman, Colin Cooper, and Alan Frieze. Min-wise independent linear permutations. *Electronic Journal of Combinatorics*, 7:R26, 2000.

[23] Paolo Boldi and Sebastiano Vigna. The webgraph framework i: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pages 595–602. ACM, 2004.

[24] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 327–336. ACM, 1998.

[25] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations (extended abstract). In *STOC '98*, pages 327–336, New York, NY, USA, 1998. ACM.

[26] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. In *WWW*, 1997.

[27] Richard Brown et al. Report to congress on server and data center energy efficiency: Public law 109-431. *Lawrence Berkeley National Laboratory*, 2008.

[28] Aniket Chakrabarti, Bortik Bandyopadhyay, and Srinivasan Parthasarathy. Improving locality sensitive hashing based similarity search and estimation for kernels. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 641–656. Springer International Publishing, 2016.

[29] Aniket Chakrabarti, Manish Marwah, and Martin Arlitt. Robust anomaly detection for large-scale sensor data. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, pages 31–40. ACM, 2016.

[30] Aniket Chakrabarti and Srinivasan Parthasarathy. Sequential hypothesis tests for adaptive locality sensitive hashing. In *Proceedings of the 24th International Conference on World Wide Web*, pages 162–172. ACM, 2015.

[31] Aniket Chakrabarti, Srinivasan Parthasarathy, and Christopher Stewart. Green-and heterogeneity-aware partitioning for data analytics. In *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on*, pages 366–371. IEEE, 2016.

[32] Aniket Chakrabarti, Srinivasan Parthasarathy, and Christopher Stewart. A pareto framework for data analytics on heterogeneous systems: Implications for green energy usage and performance. In *Parallel Processing (ICPP), 2017 46th International Conference on*. IEEE, 2017.

[33] Aniket Chakrabarti, Venu Satuluri, Atreya Srivathsan, and Srinivasan Parthasarathy. A bayesian perspective on locality sensitive hashing with extensions for kernel methods. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(2):19, 2015.

[34] Aniket Chakrabarti, Christopher Stewart, Daiyi Yang, and Rean Griffith. Zoolander: Efficient latency management in nosql stores. In *Proceedings of the Posters and Demo Track*, Middleware '12, pages 7:1–7:2, New York, NY, USA, 2012. ACM.

[35] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[36] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC '02*, 2002.

[37] K. Chatfield, V. Lempitsky, A. Vedaldi, and A. Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *British Machine Vision Conference*, 2011.

[38] D Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. Polonium: Tera-scale graph mining and inference for malware detection. In *SIAM International Conference on Data Mining*, volume 2, 2011.

[39] Jinghu Chen and Marc PC Fossorier. Near optimum universal belief propagation based decoding of low-density parity check codes. *Communications, IEEE Transactions on*, 50(3):406–414, 2002.

[40] Dazhao Cheng, Changjun Jiang, and Xiaobo Zhou. Heterogeneity-aware workload placement and migration in distributed sustainable datacenters. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 307–316. IEEE, 2014.

[41] JeeWhan Choi, Daniel Bedard, Robert J. Fowler, and Richard W. Vuduc. A roofline model of energy. In *27th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2013, Cambridge, MA, USA, May 20-24, 2013*, pages 661–672, 2013.

[42] William G Cochran. Sampling techniques. 1977. *New York: John Wiley and Sons*.

[43] A. Cockcroft and D. Sheahan. Benchmarking cassandra scalability on aws - over a million writes per second. `http://techblog.netflix.com`, November 2011.

[44] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *SOCC*, 2010.

[45] M. Datar, N. Immorlica, P. Indyk, and V.S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SOCG*, pages 253–262. ACM, 2004.

[46] J. Dean. Achieving rapid response times in large online services, 2012.

[47] J. Dean and L. Barroso. The tail at scale. 2013.

[48] J. Dean and S. Gemawat. Mapreduce: simplified data processing on large clusters. In *USENIX OSDI*, December 2004.

[49] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[50] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *ACM SOSP*, 2007.

[51] Nan Deng, Christopher Stewart, and Jing Li. Concentrating renewable energy in grid-tied datacenters. In *Sustainable Systems and Technology (ISSST), 2011 IEEE International Symposium on*, pages 1–6. IEEE, 2011.

[52] Robert H Dennard, Fritz H Gaensslen, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[53] Jeff Dike. User-mode linux.

[54] Quang Duong, Sharad Goel, Jake Hofman, and Sergei Vassilvitskii. Sharding social networks. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 223–232. ACM, 2013.

[55] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 365–376. ACM, 2011.

[56] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[57] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on*, pages 178–178. IEEE, 2004.

[58] Imola K Fodor. A survey of dimension reduction techniques. *Center for Applied Scientific Computing, Lawrence Livermore National Laboratory*, 9:1–18, 2002.

[59] William B Frakes and Ricardo Baeza-Yates. Information retrieval: data structures and algorithms. 1992.

[60] William T Freeman, Egon C Pasztor, and Owen T Carmichael. Learning low-level vision. *International journal of computer vision*, 40(1):25–47, 2000.

[61] Jesse Frey. Fixed-width sequential confidence intervals for a proportion. *The American Statistician*, 64(3), 2010.

[62] Drew Fudenberg and Jean Tirole. *Game Theory*. MIT Press, Cambridge, MA, 1991.

[63] A. Gelfond. Tripadvisor architecture - 40m visitors, 200m dynamic page views, 30tb data. `http://highscalability.com`, June 2011.

[64] Lise Getoor. *Introduction to statistical relational learning*. MIT press, 2007.

[65] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[66] MA Girshick, Frederick Mosteller, and LJ Savage. Unbiased estimates for certain binomial sampling problems with applications. In *Selected Papers of Frederick Mosteller*, pages 57–68. Springer, 2006.

[67] Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, and Steven M Seitz. Multi-view stereo for community photo collections. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[68] Íñigo Goiri, Ryan Beauchea, Kien Le, Thu D Nguyen, Md E Haque, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. Greenslot: scheduling energy consumption in green datacenters. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 20. ACM, 2011.

[69] Inigo Goiri, Ricardo Bianchini, Santosh Nagarakatte, and Thu D Nguyen. Approxhadoop: Bringing approximations to mapreduce frameworks. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 383–397. ACM, 2015.

[70] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, volume 12, page 2, 2012.

[71] Jim Gray. *Transaction Processing: Concepts and Techniques*. 1993.

[72] Douglas M Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[73] Monika Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *SIGIR*, 2006.

[74] Eben Hewitt. Cassandra: The definitive guide, 2011.

[75] S. Hsiao, L. Massa, V. Luu, and A. Gelfond. An epic tripadvisor update: Why not run on the cloud? the grand experiment. `http://highscalability.com/blog/2012/10/2/`.

[76] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX*, 2010.

[77] Ching-Lai Hwang, Abu Syed Md Masud, Sudhakar R Paidy, and Kwangsun Paul Yoon. *Multiple objective decision making, methods and applications: a state-of-the-art survey*, volume 164. Springer Berlin, 1979.

[78] Jinho Hwang and Timothy Wood. Adaptive performance-aware distributed memory caching. In *IEEE ICAC*, 2013.

[79] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, 1998.

[80] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *Computer Vision–ECCV 2008*, pages 304–317. Springer, 2008.

[81] Ke Jiang, Qichao Que, and Brian Kulis. Revisiting kernelized locality-sensitive hashing for improved large-scale image retrieval. *arXiv preprint arXiv:1411.4199*, 2014.

[82] Elsa M Jordaan and Guido F Smits. Robust outlier detection using svm regression. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 3, pages 2017–2022. IEEE, 2004.

[83] U Kang, Duen Horng Chau, and Christos Faloutsos. Mining large graphs: Algorithms, inference, and discoveries. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 243–254. IEEE, 2011.

[84] Ashish Kapoor, Zachary Horvitz, Spencer Laube, and Eric Horvitz. Airplanes aloft as a sensor network for wind forecasting. In *Proceedings of the 13th international symposium on Information processing in sensor networks*, pages 25–34. IEEE Press, 2014.

[85] Rishi Kapoor, George Porter, Malveeka Tewari, Geoffrey M. Voelker, and Amin Vahdat. Chronos: Predictable low latency for data center applications. In *ACM SOCC*, 2012.

[86] Jaimie Kelley and Christopher Stewart. Balanced and predictable networked storage. In *International Workshop on Data Center Performance*, 2013.

[87] Jaimie Kelley, Christopher Stewart, Nathaniel Morris, Devesh Tiwari, Yuxiong He, and Sameh Elnikety. Obtaining and managing answer quality for online data-intensive services. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 2(2):11, 2017.

[88] Myunghwan Kim, Roshan Sumbaly, and Sam Shah. Root cause detection in a service-oriented architecture. 2013.

[89] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[90] Michael Kozuch, Michael Ryan, Richard Gass, Steven Schlosser, David O'Hallaron, James Cipar, Elie Krevat, Julio López, Michael Stroucken, and Gregory R. Ganger. Tashi: Location-aware cluster management. In *First Workshop on Automated Control for Datacenters and Clouds (ACDC'09)*, 2009.

[91] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Outlier detection techniques. In *Tutorial at the 16th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, DC*, 2010.

[92] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(6):1092–1104, 2012.

[93] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *WWW*, 2010.

[94] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. Skewtune: mitigating skew in mapreduce applications. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 25–36. ACM, 2012.

[95] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998.

[96] Roy Levin and Yaron Kanza. Stratified-sampling over social networks using mapreduce. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 863–874. ACM, 2014.

[97] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397, 2004.

[98] D.D. Lewis, Y. Yang, T.G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *JMLR*, 5:361–397, 2004.

[99] Chao Li, Amer Qouneh, and Tao Li. iswitch: coordinating and optimizing renewable energy powered server clusters. In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, pages 512–523. IEEE, 2012.

[100] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci. Technol.*, 58:1019–1031, May 2007.

[101] Hyeontaek Lim, Bin Fan, David G. Andersen, and Michael Kaminsky. SILT: A memory-efficient, high-performance key-value store. In *ACM SOSP*, Cascais, Portugal, October 2011.

[102] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2074–2081. IEEE, 2012.

[103] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Graphlab: A new parallel framework for machine learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, Catalina Island, California, July 2010.

[104] Y. Lu, Q. Xie, G. Kilot, A. Geller, J. Larus, and A. Greenburg. Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services. In *PERFORMANCE*, 2011.

[105] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[106] Peter Matthews. A slowly mixing markov chain with implications for gibbs sampling. *Statistics & probability letters*, 17(3):231–236, 1993.

[107] Marvin McNett, Diwaker Gupta, Amin Vahdat, and Geoffrey M. Voelker. Usher: An Extensible Framework for Managing Clusters of Virtual Machines. In *Proceedings of the 21st Large Installation System Administration Conference (LISA)*, November 2007.

[108] Xiangrui Meng. Scalable simple random sampling and stratified sampling. In *ICML (3)*, pages 531–539, 2013.

[109] James Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, 209:415–446, 1909.

[110] Arif Merchant, Mustafa Uysal, Pradeep Padala, Xiaoyun Zhu, Sharad Singhal, and Kang Shin. Maestro: quality-of-service in large disk arrays. In *IEEE ICAC*, 2011.

[111] Joel C Miller and Aric Hagberg. Efficient generation of networks with given expected degrees. In *Algorithms and Models for the Web Graph*, pages 115–126. Springer, 2011.

[112] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and Analysis of Online Social Networks. In *IMC*, 2007.

[113] M. mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 2001.

[114] Jeffrey C. Mogul. Tcp offload is a dumb idea whose time has come. In *HotOS*, 2003.

[115] Yadong Mu, Jialie Shen, and Shuicheng Yan. Weakly-supervised hashing in kernel space. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3344–3351. IEEE, 2010.

[116] Yadong Mu and Shuicheng Yan. Non-metric locality-sensitive hashing. In *AAAI*, 2010.

[117] Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 467–475. Morgan Kaufmann Publishers Inc., 1999.

[118] Behnam Neyshabur and Nathan Srebro. On symmetric and asymmetric lshs for inner product search. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1926–1934, 2015.

[119] Srinivasan Parthasarathy. Efficient progressive sampling for association rules. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pages 354–361, 2002.

[120] VI Paulauskas. On the rate of convergence in the central limit theorem in certain banach spaces. *Theory of Probability & Its Applications*, 21(4):754–769, 1977.

[121] Andrew Pavlo, Carlo Curino, and Stanley Zdonik. Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 61–72. ACM, 2012.

[122] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[123] Joseph J Pfeiffer III, Sebastian Moreno, Timothy La Fond, Jennifer Neville, and Brian Gallagher. Attributed graph models: modeling network structure with correlated attributes. In *Proceedings of the 23rd international conference on World wide web*, pages 831–842. International World Wide Web Conferences Steering Committee, 2014.

[124] Heinz Prüfer. Neuer beweis eines satzes über permutationen. *Arch. Math. Phys*, 27:742–744, 1918.

[125] Erhard Rahm and Robert Marek. Analysis of dynamic load balancing strategies for parallel shared nothing database systems. In *VLDB*, pages 182–193. Citeseer, 1993.

[126] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *ACM SIGMOD Record*, volume 29, pages 427–438. ACM, 2000.

[127] D. Ravichandran, P. Pantel, and E. Hovy. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *ACL*, 2005.

[128] John A Rice. *Mathematical statistics and data analysis*. Cengage Learning, 2007.

[129] rigor.com. Why performance matters to your bottom line. `http://rigor.com/2012/09/roi-of-web-performance-infographic`.

[130] Christian Robert and George Casella. A short history of markov chain monte carlo: subjective recollections from incomplete data. *Statistical Science*, pages 102–115, 2011.

[131] Venu Satuluri and Srinivasan Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *Proceedings of the VLDB Endowment*, 5(5):430–441, 2012.

[132] Ashok Savasere, Edward Robert Omiecinski, and Shamkant B Navathe. An efficient algorithm for mining association rules in large databases. 1995.

[133] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

[134] David W Scott. Kernel density estimators. *Multivariate Density Estimation: Theory, Practice, and Visualization*, pages 125–193, 2008.

[135] Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

[136] Navin Sharma, Sean Barker, David Irwin, and Prashant Shenoy. Blink: managing server clusters on intermittent power. In *ACM ASPLOS*, March 2011.

[137] John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[138] Kai Shen, Christopher Stewart, Chuanpeng Li, and Xin Li. Reference-driven performance anomaly identification. In *ACM SIGMETRICS*, 2009.

[139] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, pages 2321–2329, 2014.

[140] David Shue, Michael Freedman, and Anees Shaikh. Performance isolation and fairness for multi-tenant cloud storage. In *USENIX OSDI*, 2012.

[141] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Descriptor learning using convex optimisation. In *Computer Vision–ECCV 2012*, pages 243–256. Springer, 2012.

[142] C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. In *EuroSys Conf.*, March 2007.

[143] C. Stewart, K. Shen, A. Iyengar, and J. Yin. Entomomodel: Understanding and avoiding performance anomaly manifestations. In *IEEE MASCOTS*, 2010.

[144] Christopher Stewart, Aniket Chakrabarti, and Rean Griffith. Zoolander: Efficiently meeting very strict, low-latency slos. In *ICAC*, volume 13, pages 265–277, 2013.

[145] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. In *IEEE/ACM Trans. Netw.*, 2003.

[146] Patrick Stuedi, Animesh Trivedi, and Bernard Metzler. Wimpy nodes with 10gbe: leveraging one-sided operations in soft-rdma to boost memcached. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, 2012.

[147] Charles Sutton, Andrew McCallum, and Khashayar Rohanimanesh. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *Journal of Machine Learning Research*, 8(Mar):693–723, 2007.

[148] Acar Tamersoy, Kevin Roundy, and Duen Horng Chau. Guilt by association: large scale malware detection by mining file-relation graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1524–1533. ACM, 2014.

[149] Shirish Tatikonda and Srinivasan Parthasarathy. Hashing tree-structured data: Methods and applications. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 429–440. IEEE, 2010.

[150] Sávyo Toledo, Danilo Melo, Guilherme Andrade, Fernando Mourão, Aniket Chakrabarti, Renato Ferreira, Srinivasan Parthasarathy, and Leonardo Rocha. D-sthark: Evaluating dynamic scheduling of tasks in hybrid simulated architectures. *Procedia Computer Science*, 80:428–438, 2016.

[151] TripAdvisor Inc. Tripadvisor reports fourth quarter and full year 2011 financial results, February 2012.

[152] Beth Trushkowsky, Peter Bodík, Armando Fox, Michael J. Franklin, Michael I. Jordan, and David A. Patterson. The scads director: Scaling a distributed storage system under stringent performance requirements. In *USENIX FAST*, 2011.

[153] Abraham Wald. *Sequential analysis*. Courier Corporation, 1973.

[154] Lizhe Wang, Gregor Von Laszewski, Jay Dayal, and Fugang Wang. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 368–377. IEEE, 2010.

[155] Ye Wang, Srinivasan Parthasarathy, and P Sadayappan. Stratification driven placement of complex data: A framework for distributed data analytics. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 709–720. IEEE, 2013.

[156] Yu Wang, Aniket Chakrabarti, David Sivakoff, and Srinivasan Parthasarathy. Fast change point detection on dynamic social networks. In *International Joint Conference of Artificial Intelligence (IJCAI)*, 2017.

[157] Yu Wang, Aniket Chakrabarti, David Sivakoff, and Srinivasan Parthasarathy. Hierarchical change point detection on dynamic networks. In *The 9th International ACM Web Science Conference 2017 (WebSci'17)*, 2017.

[158] Christian H Weiß. Sampling in data mining. *Wiley StatsRef: Statistics Reference Online*, 2015.

[159] Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Proceedings of the 14th Annual Conference on Neural Information Processing Systems*, number EPFL-CONF-161322, pages 682–688, 2001.

[160] Hao Xia, Pengcheng Wu, Steven CH Hoi, and Rong Jin. Boosting multi-kernel locality-sensitive hashing for scalable image retrieval. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 55–64. ACM, 2012.

[161] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near duplicate detection. *ACM Transactions on Database systems*, 2011.

[162] Zichen Xu, Nan Deng, Christopher Stewart, and Xiaorui Wang. Cadre: Carbon-aware data replication for geo-diverse services. In *Autonomic Computing (ICAC), 2015 IEEE International Conference on*, pages 177–186. IEEE, 2015.

[163] Xifeng Yan and Jiawei Han. Closegraph: mining closed frequent graph patterns. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 286–295. ACM, 2003.

[164] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8:236–239, 2003.

[165] Jie Yin, Derek Hao Hu, and Qiang Yang. Spatio-temporal event detection using dynamic conditional random fields. In *IJCAI*, volume 9, pages 1321–1327. Citeseer, 2009.

[166] W. Yoo, K. Larson, L. Baugh, S. Kim, and R. Campbell. Adp: Automated diagnosis of performance pathologies using hardware events. In *ACM SIGMETRICS*, 2012.

[167] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, Wei Li, et al. New algorithms for fast discovery of association rules. In *KDD*, volume 97, pages 283–286, 1997.

[168] Hao Zhang, Alexander C Berg, Michael Maire, and Jitendra Malik. Svm-knn: Discriminative nearest neighbor classification for visual category recognition. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2126–2136. IEEE, 2006.

[169] Jiong Zhang and Mohammad Zulkernine. Network intrusion detection using random forests. In *PST*. Citeseer, 2005.

[170] Yang Zhang, Nirvana Meratnia, and Paul Havinga. Outlier detection techniques for wireless sensor networks: A survey. *Communications Surveys & Tutorials, IEEE*, 12(2):159–170, 2010.

[171] Yanwei Zhang, Yefu Wang, and Xiaorui Wang. Greenware: Greening cloud-scale data centers to maximize the use of renewable energy. In *Middleware 2011*, pages 143–164. Springer, 2011.

[172] Z. Zhang, L. Cherkasova, A. Verma, and B. Loo. Automated profiling and resource management of pig programs for meeting service level objectives. In *IEEE ICAC*, September 2012.

[173] Timothy Zhu, Anshul Gandhi, Mor Harchol-Balter, and Michael A. Kozuch. Saving cash by using less cache. In *USENIX Workshop on Hot Topics in Cloud Computing*, 2012.

[174] X. Zhu and A.B. Goldberg. Introduction to semi-supervised learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 3(1):1–130, 2009.

[175] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on*, 24(5):530–536, 1978.

[176] Laurent Zwald and Gilles Blanchard. On the convergence of eigenspaces in kernel principal component analysis. In *NIPS*, 2005.