Publications of the Astronomical Society of the Pacific, 134:104501 (12pp), 2022 October https://doi.org/10.1088/1538-3873/ac8dc2




# Model-based Performance Characterization of Software Correlators for Radio Interferometer Arrays

A. J. Vázquez[1,3], P. Elosegui[1,2], C. J. Lonsdale[1], G. B. Crew[1], V. L. Fish[1], and C. A. Ruszczyk[1]
[1] Massachusetts Institute of Technology, Haystack Observatory, Westford, MA, USA; ajvazquez.teleco@gmail.com
[2] Institute of Marine Sciences, ICM-CSIC, Barcelona, Spain
*Received 2022 August 1; accepted 2022 August 30; published 2022 October 11*

## Abstract

Correlation for radio interferometer array applications, including Very Long Baseline Interferometry (VLBI), is a multidisciplinary field that traditionally involves astronomy, geodesy, signal processing, and electronic design. In recent years, however, high-performance computing has been taking over electronic design, complicating this mix with the addition of network engineering, parallel programming, and resource scheduling, among others. High-performance applications go a step further by using specialized hardware like Graphics Processing Units (GPUs) or Field Programmable Gate Arrays (FPGAs), challenging engineers to build and maintain high-performance correlators that efficiently use the available resources. Existing literature has generally benchmarked correlators through narrow comparisons on specific scenarios, and the lack of a formal performance characterization prevents a systematic comparison. This combination of ongoing increasing complexity in software correlation together with the lack of performance models in the literature motivates the development of a performance model that allows us not only to characterize existing correlators and predict their performance in different scenarios but, more importantly, to provide an understanding of the trade-offs inherent to the decisions associated with their design. In this paper, we present a model that achieves both objectives. We validate this model against benchmarking results in the literature, and provide an example for its application for improving cost-effectiveness in the usage of cloud resources.

*Unified Astronomy Thesaurus concepts:* Theoretical models (2107); Astronomy software (1855); Interferometric correlation (807); Radio astronomy (1338); Radio interferometry (1346); Distributed computing (1971)

## 1. Introduction

Radio Interferometry for astronomy and geodesy Thompson et al. (2017) is a radio technique that combines signals received with many telescopes in a computational fashion, enabling observations with high angular resolution and delay measurement precision. High resolution is achieved by using a set of sparsely distributed radio-telescopes, or "stations", pointing at the same distant radio source and combining them to form a "virtual" telescope that provides an angular resolution equivalent to that of a dish with a diameter equal to the maximum separation of telescopes. Each telescope acquires digitally sampled complex voltage signals with picosecond precision (obtained by atomic clocks); these data streams that are typically several to tens of gigabits per second (Gbps) are then ingested into a correlator to perform a Fourier-transformation and cross-multiplication among all the pairs of telescopes, followed by accumulation (summing) of the results over an interval of time known as the "accumulation period" Thompson et al. (2017). The noise portions of the signals from each of these widely separated telescopes are uncorrelated, and their complex product averages to zero. But since all the telescopes are looking in the same direction, signals from a compact, discrete radio source on the sky *are* correlated, and average to a non-zero quantity. This is an interferometric fringe pattern, whose amplitude and phase (known as the fringe visibility) are measured using the correlation processing that is the topic of this paper.

Radio interferometry applications, including VLBI, are numerous and include astronomy, where scientists are interested in imaging natural radio sources through sampling of large numbers of spatially independent interferometric visibilities as the Earth rotates. Another major application is geodesy, where regular observations of distant quasars allow for millimeter-level determination of the Earth's orientation and movement in space through precise measurements of the relative delays of the signals received at each telescope.

The computation-intensive signal correlation process has been traditionally executed by dedicated hardware correlators,

[3] Author to whom any correspondence should be addressed.

and more recently with the advent of high-performance computing by software running in the cloud or on computer clusters. The most important advantage of software correlation is scalability, which is the ability to increase performance by simply increasing the number of resources available to the correlator. Achieving scalability is not trivial however due to the multidimensional complexity of the correlation problem, which includes at least: stations, channels, and signal duration. For the sake of simplicity, in a worst-case scenario, we can assume that the computations in the correlation problem grow quadratically with the number of stations (that is, all the pairs of stations) and linearly with the channels and duration of the signal. (We will provide further detail in Section 2.1).

Different applications have different requirements not only in terms of performance but also of scalability: the VLBI Global Observing System (VGOS) mission plans on performing observations with 40 stations at 32 Gbps Niell et al. (2005), the Event Horizon Telescope (EHT) with 11 stations already (and more to come) at 64 Gbps Goddi et al. (2019), the Low Frequency Array (LOFAR) with 40 stations at 6 Gbps Broekema (2018) (initially using a dedicated correlator devised to run on IBM Blue Gene machines, currently running on a GPU-based correlator), and the Canadian Hydrogen Intensity Mapping Experiment (CHIME) Pathfinder with 256 stations at 3 Gbps Recnik et al. (2015) (using a dedicated correlator based on GPUs). For such applications, there are choices to be made for correlator cluster hardware (dedicated or cloud-based, processor architectures, interconnects, etc.) and the software that runs on that hardware. These choices can have major implications for the tradeoff between cost and performance.

The wide range of instrumental configurations leads to some questions: How can the performance of different correlators be compared? Does doubling the size of the cluster double the obtainable performance for a given scenario? How does this depend on the software architecture and implementation? Can existing software be made to run efficiently for specific target configurations? Whether or not to reuse existing software or develop new code is an important decision with significant cost implications. This question is faced by astronomy and geodesy projects and facilities during early development stages. The literature available to such projects is mostly limited to specific benchmarking (Deller et al. (2007), Keimpema et al. (2015), Deller et al. (2011), Deller & Brisken (2009), Bertarini et al. (2012), Brisken (2007), CSIRO's Australia Telescope National Facility (2009), Kettenis et al. (2009), Morgan (2008), Morgan (2010), Phillips (2009), Stagni & Nanni (2013), Wu (2015), Gill et al. (2019), Vázquez (2021)). Although some references Brisken (2007), D'Addario (2001), Brisken & Deller (2007), provide computation bounds and others Recnik et al. (2015); Wagner & Ritakari (2007) detail the data flow rates through different parts of the correlator, the lack of a systematic approach makes it difficult to draw general conclusions relevant to diverse project circumstances. Consequently, approaching performance modeling for software correlators from a formal perspective is beneficial, not only in the process of decision-making for the projects, but also in the process of designing the correlators due to their inherent complexity.

This document is organized as follows. In Section 2, we describe in detail the development of the performance model. In Section 3, we provide a preliminary validation of the model against benchmarking results from previous literature. In Section 4, we provide an application example of the model for the identification of performance bottlenecks and optimization of resources in a correlator. In Section 5, we summarize the conclusions of this work.

## 2. Headroom Model

Traditionally, the evaluation of high-performance systems has been done through benchmarking. However, considering software correlators for VLBI specifically, there is a broad range of scenarios associated with different projects, and the configuration of clusters hosting these correlators. This makes the standardization of benchmarking in software correlation a formidable task with an impractically large parameter space to cover.

On the other hand, bound-and-bottleneck models have proven to be useful in the high-level characterization of performance in the parallel-computing community for identifying bottlenecks in software running on multicore architectures Williams et al. (2009). These models, rather than providing a highly detailed characterization for specific cases, instead aim for a simple characterization of performance bounds to provide designers with useful insight on the limits and dependencies of the system.

In this work, we follow a hybrid approach that (i) leverages the general architecture of these correlators to develop a formal performance model and (ii) uses benchmarking to estimate certain parameters that are specific to each correlator. Our approach thereby minimizes benchmarking, while providing deeper insight into the tradeoffs inherent to the design of these correlators through a formal theoretical model. This allows us to estimate actual performance results since it uses measured rates associated with the cluster where the correlation software is running. We call this model a "headroom" model (a term taken from audio signal processing) since it allows one to calculate the limiting rates for data flow at each part of the system as well as estimate the level of saturation at each of them.

### 2.1. A Quick Introduction to Software Correlators

Independently of the architecture and implementation, there are four main tasks to be performed by every correlator (e.g., Recnik et al. 2015; Deller et al. 2007; Keimpema et al. 2015, Figure 1): (i) *control* or coordination of the correlation process, (ii) *data distribution* or management and distribution of the data into the nodes that will process it, (iii) *processing* or correlation of the distributed data, and (iv) *collection* or

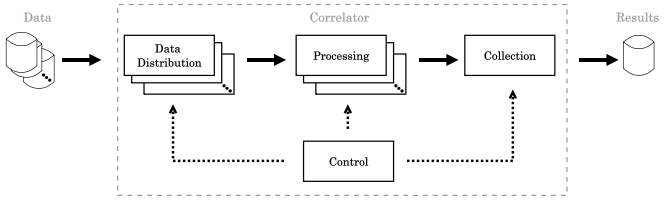


**Figure 1.** Data flow in a software correlator. Data is read and distributed by the data distribution tasks into the *processing* tasks, the results of which are then gathered by the *collection* task. The whole process is managed by the *control* task. Multiple blocks represent parallelization through multiple tasks. Blocks represent the tasks, cylinders data and results, solid arrows data flow, and dotted arrows control flow.

gathering and combination of all the results into output files suitable for further reduction and analysis.

Taking into account the high-performance computing approach, we first provide an informal description of the requirements for these tasks, which we will later develop formally. Since these tasks will be executed on a computer cluster, we will be talking about (i) traffic (associated with the throughput at the network interfaces) and (ii) computation (associated with operations done in the processors).

The control task involves both light traffic and computation loads, as it is generally associated with the processing and distribution of metadata.

Data distribution usually involves both heavy traffic and computation loads, in order to feed the rest of the chain with multiple copies of each station data stream data as quickly as possible. Note that in this document we generalize the radio array architecture to include data recording and playback steps, to logically separate telescope data acquisition and correlator input data rates. This logical separation is manifested as a physical one in the case of VLBI, historically a primary focus area for software correlation.

Software correlators typically do FX-type correlation, as described in Section 1, involving a Discrete Fourier Transform (DFT) and then a multiply-accumulate operation Deller et al. (2007); Keimpema et al. (2015), instead of the other way around (i.e., XF-type correlation). (A detailed comparison of both processing types can be found in Thompson et al. 2017). Either way, these operations are demanding both in terms of traffic and computation. Note that how data is distributed will define a trade-off between these demands, as we will show later.

Finally, collection often involves light traffic and computation loads, since even though results may be combined from many processing tasks, the fact that results are accumulated (into accumulation periods) during processing provides a large reduction in data rates. Collection can, however, become a bottleneck for cases where there are large numbers of stations, or when a wide field of view needs to be preserved, which limits the averaging that can be performed. We show the data flow among these tasks in Figure 1.

### 2.2. Scalability and Parallelization Strategies

Scalability is generally achieved through parallelization of the data distribution and the processing tasks, yet designing a correlator that is scalable is challenging due to the heavy traffic and computation loads to be accommodated on the cluster. These loads are inherent to the complexity of the correlation problem, as we now describe.

Let $S$ be the number of stations, and let $W$ be the product of the number of channels (or frequency bands) of each data stream and the number of sub-accumulation periods. These sub-accumulation periods are the divisions of the accumulation periods that are distributed among the processing tasks. Each stream requires some computation prior to being combined with other streams, and each baseline (or pair of streams in this context) also has some associated processing load. Then, we can assume that the complexity of the correlation problem is of $O((\alpha S + \beta B)W)$, where $B$ is the number of baselines computed as $B = S(S-1)/2$, and where $\alpha$ and $\beta$ depend on the splitting strategies associated with the architecture and implementation. (Lower-level details including multiple polarizations and the computation of auto-correlations are treated later in Section 2.5).

The challenge in the design of a software correlator is breaking down this complexity into distribution and processing tasks in a scalable way, requiring the selection of splitting and parallelization strategies that allow for scalable performance. Depending on how data is distributed and how processing tasks are allocated, this complexity will affect traffic and computation loads in the nodes of the cluster.

The interrelation between traffic and computation load can also be posed as an optimization problem. Consider a single sub-accumulation period to be computed for all the baselines (or pairs of stations). As we showed previously, there will be $B$ processing tasks per sub-accumulation period. In the context of mathematical graph theory, it is then easy to see that we can represent these tasks in a graph with one vertex per processing task, and one edge between every pair of tasks that have a station in common, so that the result will be an $2(S-2)$-regular graph (i.e., every vertex is connected to $2(S-2)$ vertices). Finding a splitting strategy would be equivalent to finding a balanced partitioning of this graph into subgraphs of $B_T$ nodes (representing sets of tasks to be distributed among the computation nodes). Balanced graph partitioning is an active area of research (see e.g., Andreev & Racke 2004; Pacut et al. 2021), so for the sake of simplicity, we will consider the two trivial cases: (i) $B_T = B$ (that is, each task does computations for all the baselines and thus receives streams from $S$ stations) and (ii) $B_T = 1$ (that is, each task does computations for 1 baseline, and thus receives streams from 2 stations). The number $2(S-2)$ can be understood by looking at the correlation matrix, where rows and columns represent stations, and elements of the matrix baselines; then if we pick a baseline, the ones sharing stations with it will be those in the same row plus those in the same column

minus the baseline itself and the auto-correlations (elements in the main diagonal).

The implications of selecting among these splitting strategies have not been formally addressed by previous literature, and many correlators follow the first case approach ($B_T = B$) regardless of the architecture and implementation: DiFX Deller et al. (2007), SFXC Keimpema et al. (2015), CHIME Pathfinder correlator Recnik et al. (2015), CorrelX MIT Haystack (2016), CXS338 Vázquez (2021), etc. This fact underscores the need to provide a formal model to understand such implications.

For further clarification, the next three comments describe the assumptions of our headroom model:

*Comment 1: We assume that each node (also computer or machine) of the cluster runs only one task simultaneously. Note that if we considered many data distribution tasks per node, the level of parallelization would decrease (since a single network interface would be shared among these tasks). We also dismiss other approaches that involve grouping different tasks into the same node, as this would overcomplicate this model. These extensions are left as future work.*

*Comment 2: The parallelization for data distribution can be achieved mostly by partitioning the data, but splitting the work into processing tasks is not trivial. Given that the radio array correlation problem generally requires the combination of all the pairs of streams, selecting a parallelization strategy is equivalent to partitioning a $2(S-2)$-regular graph where each vertex represents a baseline and each edge a station, aiming to minimize the number of duplicated stations between subgraphs. We will model this partition through three interrelated variables: $S_T$ the number of stations processed at each processing task, $B_T$ the number of baselines processed at each processing task (that is $B_T = S_T(S_T-1)/2$), and some factor $G_T$ representing the increase in traffic due to the distribution of data corresponding to the same station into different subgraphs (due to the overlap of vertices among them).*

*Comment 3: For the sake of simplicity, we will assume that all the stations have visibility for the complete observation time window. If this is not the case, the problem can be treated as multiple separate correlations, each with a different duration and number of stations S. More complex approaches are left as future work.*

### 2.3. Performance Metric

The first step to characterize performance is the selection of a representative metric. We use throughput $R$, which represents the amount of data processed per unit of time, for two reasons: (i) it is directly comparable with performance metrics of cluster elements (storage media, network interfaces, etc.), and (ii) it is the metric widely used in the existing literature. This throughput or datarate is calculated as $R = D_1/T_c$ with $D_1 = R_1 T_1$, where $R_1$ is the rate of the processed data for one station, $T_1$ is the recording duration of this data, and $T_c$ is the execution time of the correlation. That is, the throughput of the whole system can be computed as the total data to be processed for a single telescope over time for the complete correlation. (Refer to Comment 3 for scenarios where the data streams have different recording times.)

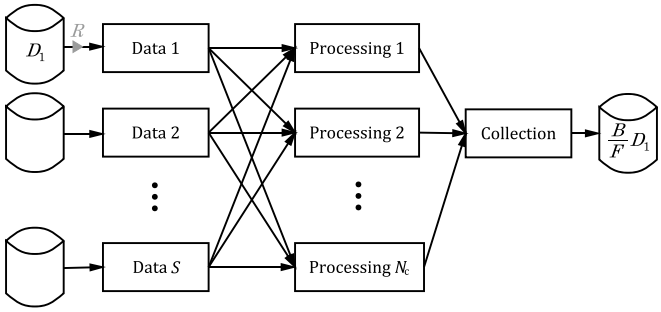


**Figure 2.** Traditional architecture for VLBI correlators. Throughput $R$ is measured relative to the data for one station. The resulting file size will be $D_1 B/F$, given an input file size of $D_1$, and where $F$ is the data reduction $F_c/F_e$.

### 2.4. Model Description

The traditional architecture for software correlators is mainly based on $S$ data nodes (or $S$ groups of $P$ nodes if data reading is parallelized) sending data to $N$ computation nodes, which after processing send the results to the collection node, as depicted in Figure 2.

According to the two basic types of load introduced previously (traffic and computation), we model the system using a network model with queues representing throughput limits inherent to certain parts of the system due to traffic (network interface limits, disk drive reading limits, etc.) and computation (data decoding, delay correction, DFT computation, etc.), as shown in Figure 3. Triangles are used to represent scaling factors between 0 and some positive real number, modeling traffic variations due to data stream splitting, gathering, and data expansion and reduction operations.

The tasks described in Section 2.1 are to be allocated to the multicore nodes of the cluster. Each of these nodes (as assumed in Comment 1) runs one single task per node but parallelizes its execution on the available processor cores. We start with data distribution and describe the chain until collection. Each data stream can be read through $P$ tasks (assuming that recorded data for a single telescope is partitioned into $P$ parts). Each of these tasks reads from disk (or playback system) at a rate $R_H$ (hard disk reading rate or playback rate), so that the data distribution limit due to data reading is:

$$R \leqslant P R_H. \tag{1}$$

Each of these data distribution tasks sends the data to the processing nodes through the network interface, and will duplicate

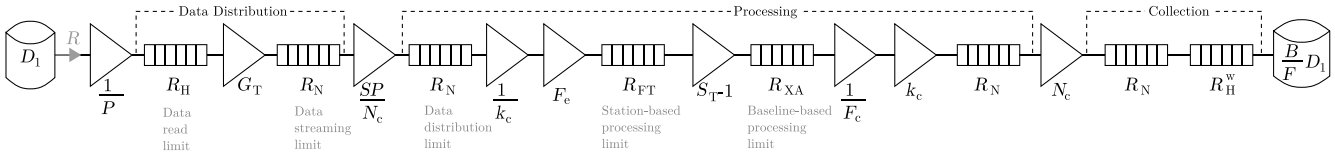


**Figure 3.** Performance model for a VLBI software correlator. Queues represent throughput limits in different parts of the system; triangles introduce multiplicative scaling factors to account for traffic splitting, gathering, expansion, or reduction. The rate $R_H$ represents the maximum rate for data reading, $R_N$ is the maximum rate for the network interface, $R_{FT}$ is the maximum rate for station-based computations, $R_{XA}$ the maximum rate for baseline-based computations, and $R_H^w$ the maximum rate for results writing. These rates are associated with the main limits of the software correlator represented at the bottom of the figure. The scaling factors $S_T$ and $G_T$ are associated with the selected parallelization strategy, $P$ with the partitioning of the input data; $N_c$ is the number of effective nodes limited by splitting, $S$ the number of stations, $k_c$ the number of cores per node; $F_e$ is associated with the expansion of coarsely sampled values into floating point precision, and $F_c$ with the reduction due to the accumulation of results.

data if $B_T < B$. From Comment 2, the traffic outgoing from each station scales by a factor of $G_T$. Taking into account the two cases introduced in Section 2.1, for generality we approximate $G_T$ as follows:

$$G_T = \frac{S-1}{S_T-1},\ 2 \leqslant S_T \leqslant S. \tag{2}$$

Therefore, taking into account the partitioning and the limit on the network interface, $R_N$, we have that:

$$R \leqslant \frac{P}{G_T} R_N. \tag{3}$$

Parallelization of the processing can be achieved by splitting the data streams into time intervals and channels, and assigning different splits to different processing tasks. It is easy to see that all the traffic outgoing from data distribution tasks equals all the incoming traffic to the processing tasks. Note that the number of effective correlation nodes, $N_c$, is limited by the nodes available for processing in the cluster but also by the splitting strategy as $N_c = \min\{N, WB/(W_T B_T)\}$, where $N$ is the number of nodes (available for processing) in the cluster and $W_T$ is the fraction of $W$ associated with each task, so that for the two cases considered:

$$N_c \leqslant \begin{cases} W/W_T, & B_T = B, \\ BW/W_T, & B_T = 1. \end{cases} \tag{4}$$

Therefore, although reducing the number of stations per task increases traffic, it also increases scalability, and thus different scenarios may call for different strategies. This motivates further work on flexible parallelization strategies.

The data distribution limit can be obtained by comparing the data outgoing from the stations (the rate $R$ scaled by a factor $G_T$, as explained above, multiplied by the number of stations $S$) with that incoming to the processing nodes (at most the network rate $R_N$ times the number of effective processing nodes $N_c$), that is, $RG_TS \leqslant N_c R_N$, and therefore:

$$R \leqslant \frac{N_c}{G_T S} R_N. \tag{5}$$

Each processing task involves station-based (e.g., DFT and delay correction) and baseline-based (e.g., multiply-accumulate) processing. Note that processing is generally done at a higher precision than sampling, so that a factor $F_e$ is introduced to account for this extension, which is roughly the precision of the processing over the bit-depth of the stream. Let $R_{FT}$ be the maximum station-based throughput for a single station per processor core and $R_{XA}$ the baseline-based processing rate for a single baseline per processor core. Note that the processing at each core is done sequentially for each of the $S_T$ stations and $B_T$ baselines, respectively, so that these computation rates represent the maximum data rate of each iteration of the (station or baseline-based) processing loop at each core.

From the limits (3) and (4) and assuming computations are distributed among the processors, we obtain the station-based and baseline-based computation limits:

$$R \leqslant \frac{N_c k_c}{G_T S} \frac{R_{FT}}{F_e}, \tag{6}$$

$$R \leqslant \frac{N_c k_c}{2B} R_{XA}. \tag{7}$$

The output rate of each correlation node is divided (compared to the input) by a factor $F_c$ which is roughly the number of DFT windows per accumulation period. This reduction will be higher if there is averaging (reduction of the number of coefficients in the resulting spectrum) after the sub-integration in the correlation nodes, so, except for very specific cases, we can obviate this limit.

These limits (1–7) correspond to those informally reported by previous literature as *I/O* (1), network (3–5), station-based processing (6), and baseline-based processing (7). (These last two are also often grouped together as CPU or computation.) Table 1 defines all the symbols used in previous sections.

The objective of this performance model is to establish a basic framework to support formal reasoning about these limits: on how to improve them in cases where they are bottlenecks, and to leverage them to optimize resources for cost-effective processing.

**Table 1**
Symbols Used in this Document

| Symbol | Description |
|---|---|
| $B$ | Number of baselines |
| $B_T$ | Number of baselines per computation task |
| $D_1$ | Total data for station 1 |
| $F_c$ | Reduction in traffic associated with accumulations, roughly number of FFTs per accumulation window (depends on accumulation window and FFT size) |
| $F_e$ | Ratio between the bit depths used in computation and in recording (recording rate is usually a small fraction, and therefore the unpacking implies an increase in traffic in the system) |
| $G_T$ | Increase in traffic due to the having $B_T < B$ |
| $k$ | Number of cores per machine |
| $k_c$ | Number of effective cores per machine (cannot be higher than $k$, limited by computation parallelization) |
| $N$ | Number of computation nodes |
| $N_c$ | Number of effective computation nodes (cannot be higher than $N$, limited by data partitioning/computation parallelization) |
| $P$ | Number of data blocks per station (in case input data is partitioned) |
| $R$ | Throughput (correlator performance, total data rate for one station divided by total execution time) |
| $R_1$ | Datarate of recorded signal for station 1 |
| $R_{FT}$ | Maximum station-based throughput for a single station per machine core (depends on the FFT size and the processor performance) |
| $R_H$ | Playback data rate (hard disk read rate) |
| $R_N$ | Network bandwidth |
| $R_W$ | Results writing data rate (hard disk write rate) |
| $R_{XA}$ | Maximum baseline-based throughput for a single station per machine core (depends on the FFT size and the processor performance) |
| $S$ | Number of stations |
| $S_T$ | Number of stations corresponding to $B_T$ |
| $T_1$ | Signal duration time for station 1 |
| $T_c$ | Correlation time |
| $W$ | Number of channels times the number of sub-accumulation windows |
| $W_T$ | Fraction of $W_T$ associated to each computation task |

### 2.5. Regarding Cluster, Experiment, and Implementation of Specific Parameters of the Model

The rates $R_H$ and $R_N$ and the number of processors per node $k_c$ can be obtained from the cluster specifications. The number of effective correlation nodes, $N_c$, is the minimum of the number of nodes available in the cluster and the number of processing tasks (that is, depends on the splitting strategy), as in Equation (4).

The rates $R_{FT}$ and $R_{XA}$ can be estimated through profiling. For the typical case $S_T = S$, $R_{FT}$ can be obtained by assuming in Equation (6) that $R$ is roughly the input data $D_1$ over the total time spent in station-based processing for a single core; and $R_{XA}$ can be obtained similarly in Equation (7) by considering the total time spent in baseline-based processing, again for a single core.

The number of stations $S$ (and baselines $B$) depends on each specific experiment. The factor $F_e$ is the precision for operations (64 or 128 bits for floating-point complex) over the number of bits per sample (from 1 to 32 Whitney et al. 2009), and the factor $F_c$ is roughly the number of DFT windows per accumulation period, as previously noted.

The number of baselines per task $B_T$ depends on the implementation. Regarding the parallelization strategy, an estimation for the factor $G_T$ has been provided in Equation (2).

### 2.6. Limitations of the Model

There are some limitations to be considered due to the assumptions made in order to provide a simple but insightful model, which we describe in this section.

Note that we provided approximations for some of the parameters for the sake of simplicity. As an example, the operational intensity of $R_{FT}$ depends on the size of the DFT, and although the model has been simplified so that $R_{FT}$ and $R_{XA}$ are independent of the number of stations, different implementations will involve different data memory schemes which could be affected by the number of stations. Although previous literature Clark et al. (2011) has addressed this topic, this level of detail would overcomplicate this document, and therefore further details like the relations between these computation rates and roofline models Williams et al. (2009) of the machines hosting the processing nodes are left as future work.

Polarizations have not been taken into account, but depending on the experiment they can be easily introduced in the model by simply considering them as stations or as an increase in the input data size, depending on whether cross-polarization correlations are required for the experiment. Regarding autocorrelations, they only affect the baseline-based processing, and they can be taken into account simply by replacing $B$ with $(B + S)$ in Equation (7).

As previously noted, this model does not consider inefficiencies due to the implementation, so the bounds provided in Section 2.4 can be considered as the best-case performance that can be provided by the system. Considering actual benchmarks, at least two components can be expected to reduce those limits: (i) some rate reduction due to fixed overheads (e.g., data decoding) and (ii) some reduction that increases with the number of nodes $N$ due to variable overhead (e.g., due to coordination of tasks).

## 3. Validation of the Model

In this section, we compare results from existing literature with the estimations that the model yields based on the reported configuration, providing a first step in assessing the utility of the presented model.

### 3.1. Scalability Benchmarking

Scalability benchmarking in software correlators usually reduces to measuring throughput in two-dimensions: number of stations $S$, and number of correlation nodes $N$. Here, we consider

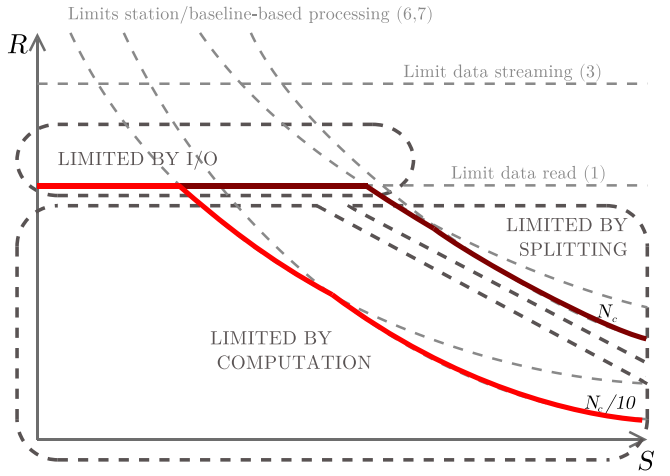


**Figure 4.** Performance regions in a scalability benchmarking plot showing throughput, $R$, vs. the number of stations, $S$. As the number of nodes in the cluster increases (curves from light-red toward dark-red), the computation limits rise. Once the splitting limit is reached, depending on the number of stations, throughput may be limited by I/O or by splitting.

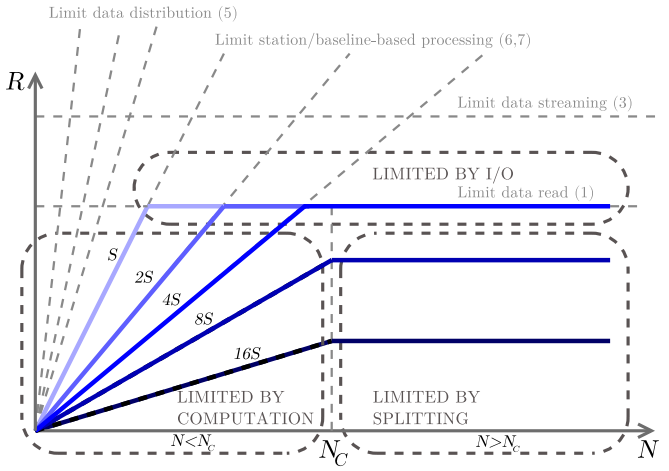


**Figure 5.** Performance regions in a scalability benchmarking plot showing throughput, $R$, vs. the number of nodes, $N$. As the number of stations increases (curves from light-blue toward dark-blue), the slope of the computation limit decreases. Note that if performance is limited by splitting, naive solutions like increasing the rates of the playback units, disks, or network interfaces will not increase throughput.

only the case where each processing task processes all the baselines ($B_T = B$), which is the approach taken by widely used correlators like DiFX Deller et al. (2007), SFXC Keimpema et al. (2015), or the CHIME correlator Recnik et al. (2015).

We provide in Figures 4 and 5 simple graphic representations to show how these limits relate to the scalability benchmarks, showing the three regions corresponding to the main limits described in Section 2.4: *I/O* [Equation (1)], computation [Equations (6), and (7)] and splitting [if $N \geqslant N_c$, Equation (4)]. For illustrative purposes, we consider a typical case where the reading rate dominates the data streaming rate ($R_H < R_N$), and the computation rates (6) and (7) dominate the data distribution rate, Equation (5), so that we can dismiss network limits in these representations.

Benchmarking could also be represented in three-dimensions, with the x-coordinate being the number of stations $S$, the y-coordinate the number of computation nodes $N$, and the z-coordinate measured performance $R$; then, a benchmarking graph would correspond to a plane (either varying $S$ with fixed $N$, like in Figure 4, or vice-versa, like in Figure 5).

### 3.2. Estimation of the Computation Rates

Although available benchmarking reports (see list in Section 1) usually provide some details regarding the specifications of the hardware running the correlator, to the best of our knowledge very few of them provide profiling information with timing results for their code.

Reference Wagner & Ritakari (2007) provides timing information for the DiFX correlator (for a DFT size of 1024), reporting 21 s spent in the routine corresponding to the station and baseline-based processing in the correlation node (out of 49 s total execution time from the list in Wagner & Ritakari (2007), page 8) for an input data of 160 MB (corresponding to 4 stations with 40 MB per station) for a single-core Intel Pentium 4 at 3.0 GHz. Following the method presented in Section 2.5, the rate 160 MB / 21 s would correspond in the model (Figure 3) to the rate measured just before the scaling-block $F_e$ and, therefore, $R_{FT}/F_e \approx 0.059$ Gbps. For an Intel Dual Core a total execution time of 15 s is reported, which, assuming linear scaling, would correspond to $R_{FT}/F_e \approx 0.097$ Gbps for a single core. We take the average of both values as a rough estimation for the computation rate, and thus we assume that $R_{FT}/F_e \approx 0.08$ Gbps.

However, it has been shown that this limit is strongly dependent on the size of the DFT, and it can drop by a factor of 10 for very long sizes Van Straten & Bailes (2011). If we consider another scenario Gill et al. (2019) with a DFT of 262144, and solve Equation (6) for $R_{FT}/F_e$ with $S = 2$, we obtain a best-case value that is one-fourth of the original, $R_{FT}/F_e \approx 0.02$. As the number of stations increases, baseline-based processing becomes more limiting than station-based processing (Gill et al. 2019 indicate that "the nonlinear term begins to dominate at large $S$ with a crossover point at $S \approx 11''$). Therefore, for this case, we estimate $R_{XA}$ in a similar way solving Equation (7) for $S = 20$.

The DFT size (or the number of spectral channels in the visibilities) depends on the experiment; as an example, whereas VGOS Barrett et al. (2019) uses only 128 channels, the EHT Gill et al. (2019) may have as many as 262144. We will use the initially computed rate $R_{FT}/F_e \approx 0.08$ Gbps in all the comparisons that we present in the following section, except for the last two results (Figures 8 and 11, that employ the

longer DFT size, 262144), where we will assume the reduced rate $R_{FT}/F_e \approx 0.02$.

A more precise characterization of these rates would provide more accurate bounds, but such characterization is left as future work.

### 3.3. Results for the Distributed FX Correlator (DiFX)

The Swinburne University of Technology's DiFX correlator Deller et al. (2007); DiFX Software Code (2016) is a widely used software correlator for VLBI. This system, written in C++, was initially devised to run on a commodity-computer cluster (a.k.a Beowulf cluster) with the Message Passing Interface (MPI) Barney (2015), and using highly-optimized-processor, proprietary libraries for vector calculations. Its architecture is defined by four kinds of entities that correspond to the tasks described in Section 2.1, except that the control and collection are performed at the same node. In this section, we will compare the results presented in Deller & Brisken (2009); Morgan (2010); Phillips (2009); Wu (2015) and Gill et al. (2019) with the bounds estimated from the model.

Results in Deller & Brisken (2009) are for a cluster of 5 nodes, each with two octa-core processors (in total 80 computer processing units) connected through Gigabit Ethernet, so that we assume $N_c = 5$, $k_c = 16$ and $R_N = 1$ Gbps. Figure 2 in Deller & Brisken (2009) shows the ratio between correlated time and observe time. From Section 2.3 it is easy to see that throughput can be computed as the product of the stream rate and the inverse of that ratio. Note that Deller & Brisken (2009) shows a boundary attributed to the capacity of the network interconnection. We use the estimate $R_{FT}/F_e \approx$ 0.08 Gbps from Section 3.2 given that this parameter is not available in the reference. We plot these results in Figure 6 along with the theoretical bounds estimated from the model. As described in previous sections, the curves with the lowest values define the limits for performance. In this case, the data streaming limit (output network interface of the data distribution nodes) limits performance until roughly $S = 4$ stations, where this limit intersects the data distribution limit; and for more stations, performance drops under this curve, limited by the input network interfaces of the processing nodes.

Reference Phillips (2009) presents benchmarking results varying the number of nodes for different numbers of cores for $S = 6$. We take the results for $k_c = 8$, assume a network of $R_N = 1$ Gbps, and use the same estimate for the computation rate as in the previous case. We plot these results with the estimated bounds from the model in Figure 9. The model predicts that performance increases linearly with the number of nodes $N$ (limited by station-based processing) and stops scaling where the station-based processing and the data streaming limits cut, at roughly $N = 9$, remaining constant for higher values of $N$ (limited by the data streaming limit).

We follow the same procedure for the results presented in Wu (2015) and Morgan (2010), and display their results along

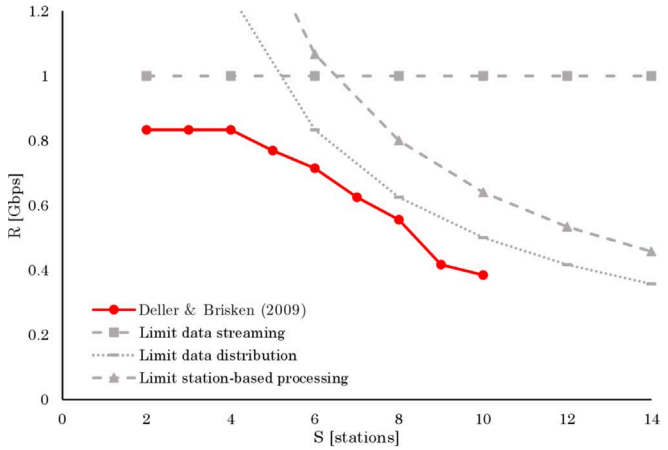


**Figure 6.** Comparison of benchmarking results for DiFX presented in Deller & Brisken (2009) in 2009 with the throughput boundaries estimated with the model.

with the estimated bounds from the model in Figures 7 and 10 respectively. For Wu (2015) we consider $N_c = 20$, $k_c = 10$ (number of cores reported in Intel® Xeon® Processor 2015 for the processor used in Wu 2015) and for Morgan (2010) we take the results for $S = 4$ for their 10 Gbps interconnected cluster ($k_c = 8$ and $R_N = 10$ Gbps from Morgan 2010). In both cases the model shows that performance is limited by station-based processing, that is the lowest curve visible in the plotted sections, and the available data does not allow for the observation of intersections with other limits. The differences between the model's estimations and the measurements could be related to the selected station-based computation rate (the same value estimated in Section 3.2 is applied to Figures 6–10).

The most recent benchmark presented in this paper for DiFX is taken from Gill et al. (2019). In that study they consider laboratory-generated data for a number of stations that ranges between 2 and 20, testing vertical scaling (increasing the number of virtual cores for a single machine) between 16 and 96 in the cloud (Google GCP Google 2022). The network has a limit of $R_N = 6$ Gbps, since there is only one machine we have that $N = 1$, and we show the results for $k_c = 16$, with the number of stations ($S$) varying between 2 and 5. This experiment considers two polarizations for each station, so as explained in Section 2.5 this is equivalent to considering twice the number of stations for computing the limits. Again in this case performance is limited by station-based processing (the lowest curve) and the model predicts a stronger drop in performance at roughly $S = 12$, where the theoretical station and baseline-based limits intersect, although in this case, the measured performance drops a bit earlier at roughly $S = 10$.

### 3.4. Results for CorrelX on Spark (CXS)

MIT Haystack's CorrelX MIT Haystack (2016), and the recent fork CXS338 Vázquez (2021), are alpha version

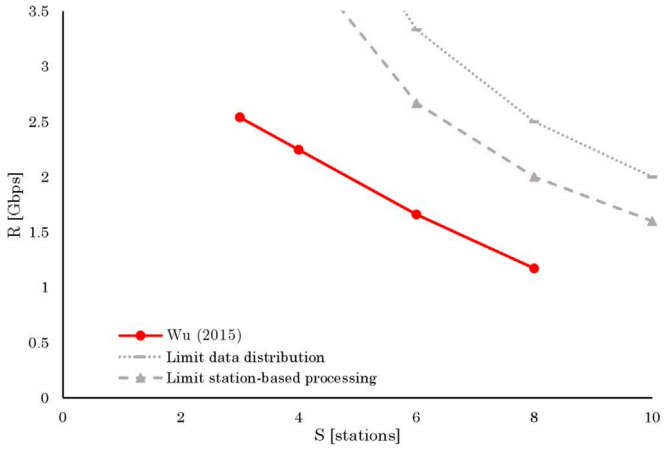


**Figure 7.** Comparison of benchmarking results for DiFX presented in Wu (2015) in 2015 with the throughput boundaries estimated with the model.

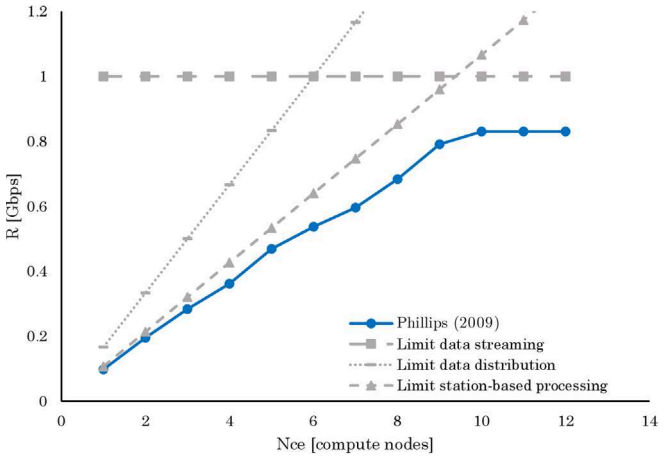


**Figure 9.** Comparison of benchmarking results for DiFX presented in Phillips (2009) in 2009 with the throughput boundaries estimated with the model.

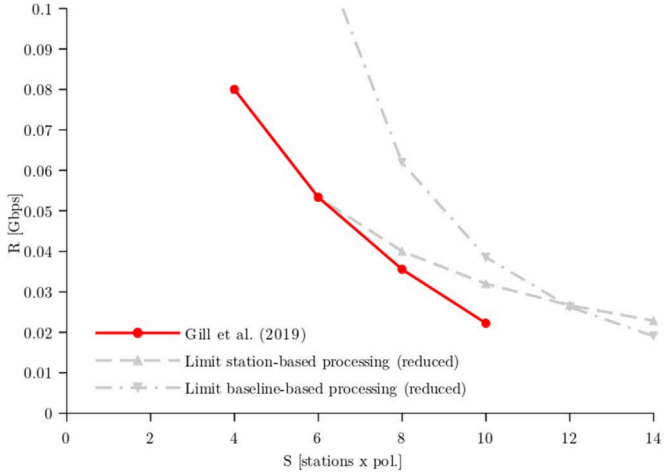


**Figure 8.** Comparison of benchmarking results for DiFX presented in Gill et al. (2019) in 2019 with the throughput boundaries estimated with the model. In this case the reduced station-based limit is considered due to the DFT size.

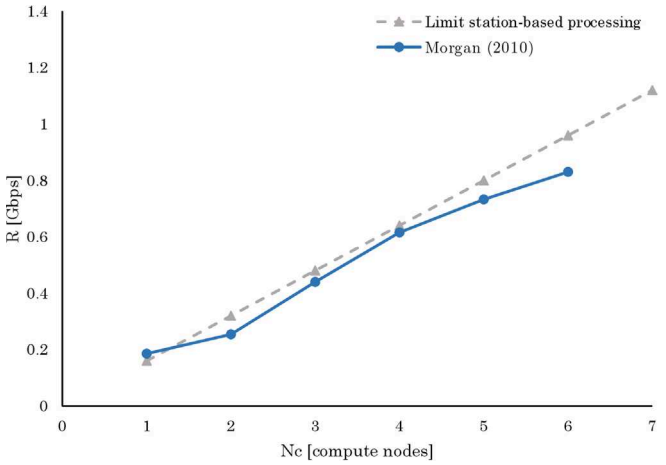


**Figure 10.** Comparison of benchmarking results for DiFX presented in Morgan (2010) in 2010 with the throughput boundaries estimated with the model.

software correlators designed to run in cloud environments, specifically the Apache ecosystem: CorrelX on Hadoop Apache (2022), and CXS on Spark Apache (2022). Both correlators are written in Python, and released under an MIT license. Unlike DiFX, these correlators do not require a careful configuration of the topology of the system since the load is distributed among the available nodes by the parallelization framework. Relying on the framework simplifies the planning of the cluster, and will allow the system to scale horizontally much more easily, but it could also decrease performance, and this could be challenging for the presented model.

Given that CXS has not yet undergone meaningful performance optimization, one expects lower performance than with DiFX. It was recently reported to run at about one-fourth of the speed reached by DiFX for a recent experiment Vázquez (2021). Based on this, we will assume that for the results that we show for CXS, the station-based limit will be determined by one-fourth of the rate $R_{FT}/F_e$ considered for DiFX in the previous section (Figure 8). Reference Vázquez (2021) (Section 5.5.3) presents benchmarking results running on the cloud (Amazon EMR Amazon Web Services 2022) on machines ($N_c$ between 1 and 8) with $k_c = 2$ cores, with data that follows the description of the data set used in Gill et al. (2019) but with a reduction in size (described in Section 5.5.1 in Vázquez 2021). As for the results in Figure 8, the number of stations $S = 2$ was adjusted to 4 to account for the dual-polarizations. We show the results in Figure 11, where besides the station-based limit displayed in other figures (with $R_{FT}/F_e \approx 0.02$), we also show the reduced station-based limit ($R_{FT}/F_e \approx 0.005$). In this case again, performance is limited by station-based processing, and it is interesting to note that there is a performance plateau starting at 7 nodes. This is because, unlike DiFX, CXS does not

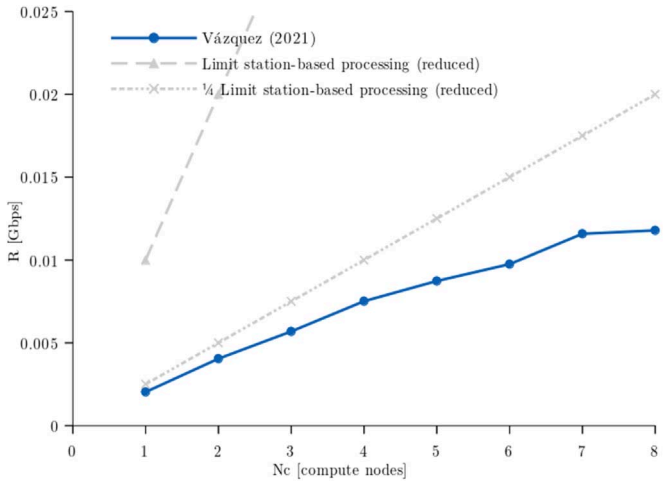


**Figure 11.** Comparison of benchmarking results for CXS presented in Vázquez (2021) in 2021 with the throughput boundaries estimated with the model. In this case, the reduced station-based limit is considered due to the DFT size.

incorporate yet the ability to perform sub-accumulation-window calculations Vázquez (2021) (Section 7.2), and therefore performance scales in a stepped way following the inverse of the ceiling function of the ratio between the number of tasks and the total number of cores (as described in Vázquez 2021 Section 5.5.3).

The reasons to add CXS into the comparison, despite this difference in performance and the limited results available, are twofold: (i) the potential of the project, written in a high-level popular language such as Python and running on a popular cloud framework that natively exploits cloud-based parallelization systems and (ii) the architecture of the system, focused on simplicity and scalability (correlation is performed in two stages with batches of tasks distributed on all the available computation infrastructure as opposed to "streaming" correlators that follow more strictly the architecture presented in Figure 2). This last reason is especially interesting, as this shows that the model is valid for different correlator architectures.

### *3.5. Discussion*

In the previous two subsections, we have presented multiple examples with benchmarking results from existing literature, and have compared measured and predicted performance with the model presented in this paper (Figures 6–11). In all the studied cases the model correctly reproduced the shape of the performance curve and shed light on the performance bottleneck that applied in each case.

We have shown that the model fits the measurements despite the variability of experiment configurations, for a reference correlator widely used by the radio astronomy community (Figures 6–10). We added a comparison with an alternative alpha-version correlator with lower performance, showing that the model supports performance comparisons between different correlators (Figures 8 and 11). Using this model and adjusting parameters appropriately, it is anticipated that performance comparison for different correlators based on different underlying technologies and architectures, and with different experiment configurations, will be possible at low cost.

Further work will help in characterizing the model parameters for specific correlators. This modeling approach represents a step forward beyond the existing literature on performance benchmarking, traditionally limited to curve fitting (finding the transition in computation limits) and conjecturing (trying to explain performance regions).

## 4. Application Example: Bottleneck Identification and Cost Optimization

In this section, we provide an example to show the applicability of the model with two objectives: (i) identifying the performance bottleneck of the system and (ii) optimizing the cost of a cloud correlation.

We will consider the scenario corresponding to the benchmark shown in Figure 9, taken from Deller & Brisken (2009), but with four different variations changing the number of stations $S$, the number of nodes $N$, and the number of cores per node $k_c$. This is shown in Figure 12, where we show the data rate at all parts of the correlator. In this representation, the scaling factors (triangles) define the load distribution, and the throughput limits (queues) define the headroom for this distribution. As explained previously, performance is measured at the input of the system (first column in the diagram). For these scenarios, the limits defined by the queues will be constant but the slopes defined by the scaling factors will change, as explained in previous sections.

Starting with the scenario $S = 4$, $N = 5$, and $k_c = 8$ and duplicating the number of stations (switching from the gray to the green distributions of loads) switches the system from being limited by data-streaming to be limited by data-distribution (as shown in Figure 9).

Now let us consider a case with more stations and higher capacity machines: $S = 64$, $N = 8$, and $k_c = 32$, represented in Figure 12 in blue. In this case, performance is limited by baseline-based processing. If at this point we continue to increase the number of nodes, for example multiplying it by 8x (red curve in Figure 12), this switches this limit to data-collection; at this point the collection node's input network is saturated.

It is easy to see for this last case that it is possible to reach the same performance with a smaller number of nodes. This reduction would imply a variation in the slopes of the fifth and thirteenth sections of the throughput representation in Figure 12 (scaling factors that depend on $N_c$, joining the data distribution, processing, and collection blocks) until the system reaches the previous limit. Although not represented in the graph, roughly halving the number of nodes (setting $N_c = 28$) transitions the system to be limited by baseline-based processing, keeping the same throughput but using only part of the available computing resources.

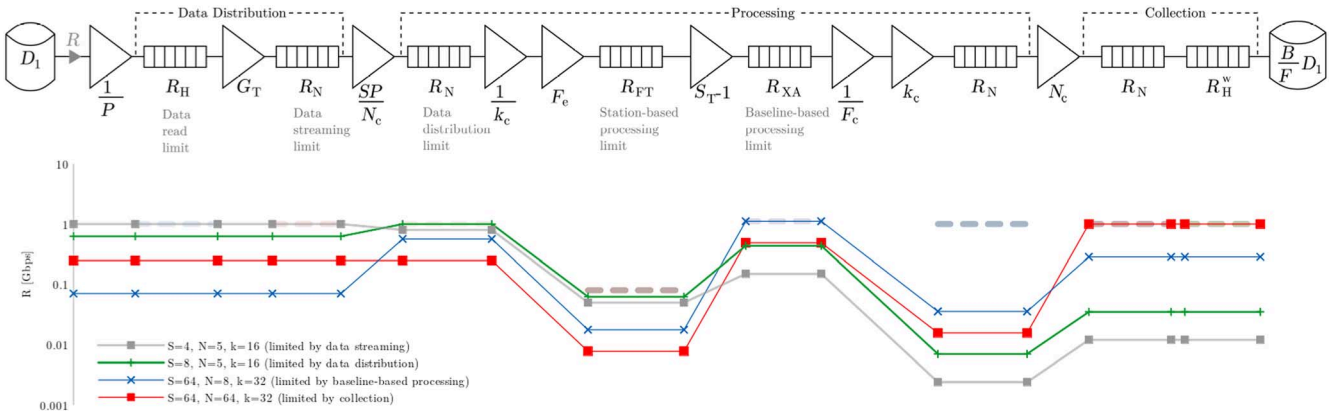


**Figure 12.** Representation of the throughput (bottom plot) at different parts of the correlator (top plot), where the bounds obtained from the model (1–7) are displayed as dashed lines following the same order as in the legend. The throughput $R$ of the system corresponds to the value in the first column of the graph. The model representation from Figure 3 is repeated on top of this figure for easy identification of the different parts of the correlator in the graph. The graph (bottom plot) corresponds to a scenario similar to the one presented in Figure 9 with four different variations: (i) $S = 4$ stations, $N = 5$ nodes and $k_c = 16$ cores in gray –limited by data streaming–, (ii) a case that duplicates the number of stations ($S$) in green –limited by data distribution–, *(iii)* a case with $S = 64$ stations, $N = 8$ nodes and $k_c = 32$ cores in blue –limited by baseline-based processing–, and another case *(iv)* that multiplies by 8x the number of nodes in red –limited by data collection–. This figure has been generated using a simple implementation of the performance model equations described in Section 2.4 and illustrates how the headroom in each part of the correlator can be represented visually.

This problem is relevant both to local and cloud-based cluster environments, where cost is generally a concern Gill et al. (2019), and it could be the case that depending on the type of experiment to be processed, a cluster composed of low-performance machines is able to do the job in the same time, for a lower price. In this case, the model could be used to find the machines with the lowest specifications that support the desired processing rate.

Referring back to the variability in correlator architectures introduced at the end of Section 3.4, it is worth noting that commercial cloud infrastructure pricing varies with the type of service, and the pricing for Elastic MapReduce (EMR) services (specifically for Apache Spark) is roughly one-fourth of those for the general purpose machines (EC2) Amazon Web Services (2022). The EMR service supports running correlators like CXS338, and even if an optimized version of CXS338 has inherently lower performance than the standard (Section 3.4), it is likely to be lower in cost due to the reduction in cost per machine, resulting in a better performance-cost ratio.

These simple examples show that, in practical scenarios, deep knowledge of the performance of the correlator allows the system designers and operators to make better decisions about the sizing of the cluster and tuning of the correlator, therefore allowing them to optimize processing times for higher performance and better cost-effectiveness.

## 5. Conclusion

We have presented the first formal performance characterization of radio astronomy correlators. Although we have focused on software correlators running on CPU clusters, this modeling approach is readily extensible to correlators that use hardware accelerators like FPGAs and GPUs as long as they follow a similar processing architecture. This work represents a step forward beyond conventional wisdom and informal reasoning from previous literature.

We have tested the model with a widely used software correlator from the VLBI community, and an alpha version of a recently released cloud correlator, by comparing benchmarking results from previous literature with the throughput limits estimated by the model, showing promising results with only a few parameters to feed the model. The model has been kept simple enough to be insightful, so that bottlenecks along the system can be identified without the need for extensive benchmarking. Compared to previous work, the model provides estimates of performance and scalability for the general case, rather than reducing the results to the specific benchmarked scenarios.

We have also shown the importance of performance modeling for better cluster/cloud planning and cost-effectiveness, presenting an example of how to use the model to understand performance bottlenecks for different configurations.

We consider this work as the first steps in modeling software correlators in radio astronomy, which we believe will help to improve current systems, but also will provide better architectures and designs for the next-generation systems.

The work by MIT Haystack Observatory was supported under NASA contracts NNG15HZ35C and 80GSFC20C0078, and MSIP award AST-2034306. The authors would like to thank V. Pankratius for providing feedback on an early draft of this paper, and also an anonymous reviewer whose constructive suggestions helped improve the manuscript.

## ORCID iDs

A. J. Vázquez https://orcid.org/0000-0002-3437-6896
P. Elosegui https://orcid.org/0000-0002-4120-7855
C. J. Lonsdale https://orcid.org/0000-0003-4062-4654
G. B. Crew https://orcid.org/0000-0002-2079-3189
V. L. Fish https://orcid.org/0000-0002-7128-9345
C. A. Ruszczyk https://orcid.org/0000-0001-7278-9707

## References

Amazon Web Services 2022, Amazon EMR pricing, https://aws.amazon.com/emr/pricing/ (Accessed February, 2022)
Amazon Web Services 2022, Amazon EMR, https://aws.amazon.com/emr/ (Accessed February, 2022)
Andreev, K., & Racke, H. 2004, Balanced Graph Partitioning (Symposium on Parallelism in Algorithms and Architectures) (Berlin: Springer)
Apache 2022, Hadoop, https://hadoop.apache.org/ (Accessed February, 2022)
Apache 2022, Spark, https://spark.apache.org/ (Accessed February, 2022)
Barney, B. 2015, Message Passing Interface (MPI) Lawrence Livermore National Laboratory, https://computing.llnl.gov/tutorials/mpi/
Barrett, J., et al. 2019, VGOS Data Processing Manual. MIT Haystack Observatory, www.haystack.mit.edu/wp-content/uploads/2020/07/docs_hops_000_vgos-data-processing.pdf
Bertarini, A., Alef, W., Müskens, A., et al. 2012, DiFX Correlator at Bonn. VII IVS General Meeting "Launching the Next Generation IVS Network", IVS, www.oan.es/gm2012/pdf/oral_id_25.pdf
Brisken, W. 2007, VLBA Sensitivity Upgrade MEMO 17: Benchmarking DiFX on the Mark5A Cluster and Justification for an Upgrade. National Radio Astronomy Observatory, https://library.nrao.edu/public/memos/vlba/up/VLBASU_17.pdf
Brisken, W., & Deller, A. 2007, The Cost of Software Correlation. VLBA Sensitivity Upgrade MEMO 16, NRAO, https://library.nrao.edu/public/memos/vlba/up/VLBASU_16.pdf
Broekema, P. C. 2018, Cobalt: A GPU-based correlator and beamformer for LOFAR. Astronomy and Computing, Vol. 23 (Amsterdam: Elsevier), 180
Clark, M. A., Plante, P. L., & Greenhill, L. J. 2011, Int. J. High Performance Comput. Appl., 27, 178
CSIRO's Australia Telescope National Facility 2009, DiFX Performance Testing, www.atnf.csiro.au/vlbi/evlbi/memos/DiFX_Performance_Testing.pdf
D'Addario, L. 2001, Correlators: General Design Considerations. ATA Memo 24. SETI, www.seti.org/sites/default/files/ATA-memo-series/memo24.pdf
Deller, A., & Brisken, W. 2009, in "Software" correlators in radio astronomy (Bellingham, WA: SPIE Newsroom)
Deller, A. T., Brisken, W. F., Phillips, C. J., et al. 2011, PASP, 123, 275
Deller, A. T., Tingay, S. J., Bailes, M., & West, C. 2007, PASP, 119, 318
DiFX Software Code 2019, Revision 7045, https://svn.atnf.csiro.au/difx (Accessed January, 2016)
Gill, A., Blackburn, L., Roshanineshat, A., et al. 2019, PASP, 131, 124501
Goddi, C., Crew, G., Impellizzeri, V., et al. 2019, The Messenger, 177, 25
Google 2022, Google Cloud Platform, https://cloud.google.com/gcp
Intel® Xeon® Processor E52660 v3 2015, https://svn.atnf.csiro.au/difx (Accessed May, 2016)
Keimpema, A., et al. 2015, The SFXC software correlator for Very Long Baseline Interferometry: Algorithms and Implementation Exp Astron, Vol. 39 (Berlin: Springer), 259
Kettenis, M., Keimpema, A., Small, D., et al. 2009, eVLBI with the SFXC software correlator. NEXPReS, JIVE, www.oan.es/expres09/pdf/kettenis.pdf
MIT Haystack 2016, CorrelX: A Cloud-Based Software Correlator for Very Long Baseline Interferometry VLBI, https://github.com/MITHaystack/CorrelX
Morgan, J. S. 2008, The DiFX Software Correlator at IRA. Istituto di Radiostronomia, IRAINAF. IRA 419/08, www.ira.inaf.it/Library/rapp-int/419-08.pdf
Morgan, J. S. 2010, PhD Thesis, University of Bologna
Niell, A., Whitney, A., Petrachenko, B., et al. 2005, VLBI2010: Current and Future Requirements for Geodetic VLBI System Int. Rep., IVS Directing Board
Pacut, M., Parham, M., Schmid, S., et al. 2021, Optimal Online Balanced Graph Partitioning, in IEEE Conf. Computer Communications (New York: IEEE),
Phillips, C. 2009, DiFX Performance Testing. eVLBI Project Scientist, https://slideplayer.com/slide/8460062/
Recnik, A., Bandura, K., Denman, N., et al. 2015, An efficient real-time data pipeline for the CHIME Pathfinder radio telescope X-engine, in 2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP) (Toronto, ON)(IEEE), 57
Stagni, M., & Nanni, M. 2013, DiFX 2.x software correlator at IRA An installation and operation manual. IRA 471/13, www.ira.inaf.it/Library/rapp-int/471-13.pdf
Thompson, A. R., et al. 2017, Interferometry and Synthesis in Radio Astronomy (3rd edn.; Berlin: Springer)
Van Straten, W., & Bailes, M. 2011, PASA, 28, 1
Vázquez, A. J. 2021, CXS338: High Performance Correlation for VLBI in the Cloud with Python. From Vision to Instrument: Designing the Next-Generation EHT to Transform Black Hole Science
Vázquez, A. J. 2021, Proof of concept of VLBI correlator based on Spark, http://e-spacio.uned.es/fez/view/bibliuned:master-ETSInformatica-ICD-Ajvazquez
Vázquez, A. J. 2021, CXS338: a high performance VLBI correlator written in Python, based on Apache Spark, https://github.com/ajvazquez/CXS338 (Accessed August, 2022)
Wagner, J., & Ritakari, J. 2007, A Playstation3-based software correlator for eVLBI. VI eVLBI, www3.mpifr-bonn.mpg.de/div/vlbi/6th_evlbi/eVLBI_presentations/JWagner.pdf
Whitney, A., Kettenis, M., Phillips, C., & Sekido, M. 2009, Proc. Sci., 082, 9
Williams, S., Waterman, A., & Patterson, D. 2009, Comm. ACM, 52, 65
Wu, J. 2015, in SHAO-DiFX correlator. VIII EAVN workshop (Shanghai: Shanghai Astronomical Observatory, CAS)