Human Factors in Design, Engineering, and Computing, Vol. 159, 2024, 916–926
https://doi.org/10.54941/ahfe1005658

HPCMod: Agent-Based Modeling Framework for Modeling Users on High-Performance Computing Resources

Nikolay Simakov

SUNY University at Buffalo, Center for Computational Research, Buffalo, FL 14203, USA

ABSTRACT

High-performance computing (HPC) resources are used for compute-demanding calculations in various fields of science and engineering. They are large computational facilities utilized by many users simultaneously. High utilization often leads to high waiting times. Simulating users' behavior on such a system can help with future system design, develop user interventions, and ultimately improve the user's experience and resource utilization. Here, we present HPCMod, an Agent-Based Modeling Framework for Modeling Users on HPC Resources. The key concept of the framework is the representation of the user's computational needs: the user project is represented as a collection of possibly dependent compute tasks. Each task can be executed as a single compute job or a series of jobs, depending on the task size. Some tasks can be too big to be executed in one chunk; such a situation often occurs during molecular dynamics simulation. There are multiple ways in which tasks can be split into jobs, and users will make their decisions based on previous experience, application parallel scalability, and available resources. For example, a user's compute task requires 32 node hours; it can be executed in multiple ways: a single 32-hour job on one node, two sequential 16-hour jobs on one node, one 16-hour job on two nodes, and so on. In the HPCMod, we implemented three models: 1) historical replay of compute jobs, 2) simulation of reconstituted compute tasks using historical job sizes, and 3) adaptive compute tasks splitting where users can modify jobs parameters given available resources till the execution of the next job in line. The framework was tested on a ten-node test system and a larger 1,736-node system modeled after a portion of TACC Stampede-2. The HPC resource model implements a first in first out (FIFO) scheduler with backfill scheduling. The initial results showed that on a tiny system, adaptive task-splitting is beneficial for the user but leads to a larger number of jobs. On a large system, the adaptive task-splitting was also very beneficial, decreasing waiting times for users using this strategy almost two times; however, other users got a 5% increase in their wait time. Further investigation is needed as the current task reconstitution algorithm is deterministic and does not allow quantification of job recombination uncertainties. The Julia-based implementation is fast: five years of historic workflow consisting of a million jobs and a one-hour stepping took around three minutes.

Keywords: High-performance computing, User interaction, Scheduling, Agent-based modelling, Workflow

INTRODUCTION

Cyber-infrastructure and high-performance computing (HPC) are increasingly important in our digital society. It is hard to underestimate the role of computational resources in numerous fields of science and engineering. Thus, it is unsurprising that HPC resources are often overutilized, resulting in high wait times and long queues. HPC resources are expensive and require high electrical power. Given that, it is crucial that such resources run optimally. Various optimizations and user interventions can improve utilization, allowing more calculations on existing facilities, enhancing user experience, and improving overall energy efficiency. HPC resource simulators are essential tools for such optimizations. This work presents our initial implementation of HPCMod, an Agent-Based Modeling (ABM) Framework for Modeling Users on High-Performance Computing Resources, and its usage to study how the knowledge of available resources can affect resource utilization. HPCMod stands for **HPC** resources **Mod**eling framework.

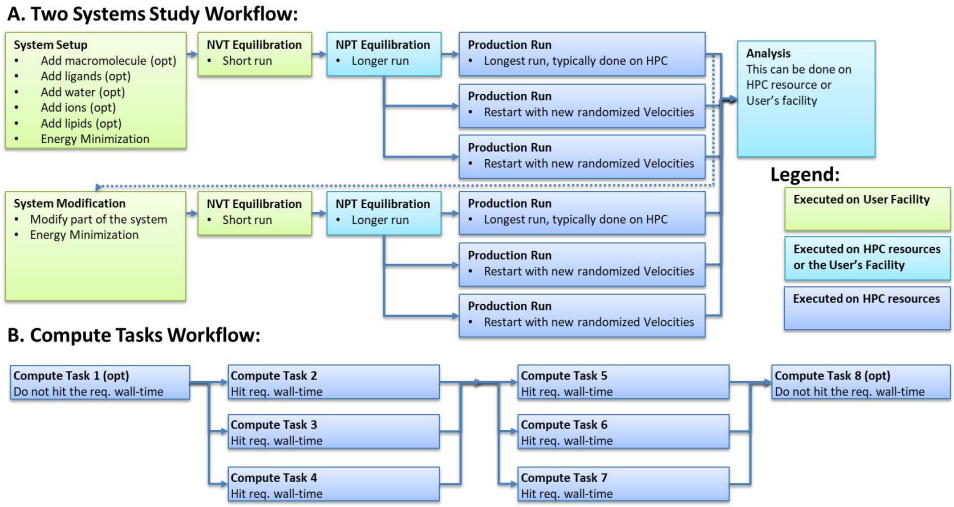

Figure 1: Representing a molecular dynamics study (A) as compute task workflow (B).

HPC resources typically consist of hundreds to thousands of compute nodes. Each node can have tens to hundreds of CPU cores and multiple GPUs. These resources are shared by multiple users, and their joint work is done through batch computing: the user writes a job script that specifies the needed resources (for example, two compute nodes for two days) and script work to be done, then the user submit it to the resource manager (also known as scheduler), and when resources become available the resource manager allocate resources and execute the job script. The job can be finished before the requested wall time, or the scheduler can terminate it if the job exceeds it. The time between job submission and job script execution start time is called wait time. It can be hours to days on busy systems.

In the past, our group developed Slurm Simulator (Simakov et al., 2018a, 2018b; Simakov and Deleon, 2024), which is based on Slurm, the most widely used HPC resource manager (Yoo et al., 2003). Built within the Slurm code, the simulator can be used for very detailed and real resource-specific simulation; however, such detailization comes with a price of relatively low simulation speed. Our latest version allows the simulation of a 200-node system with 500 times acceleration (that is, a month-long workload can be simulated in 1.5 hours). This can be problematic for multi-year simulation of larger resources and slow down the development of the user model (Multiple simulation is required due to stochastic processes). The HPCMod is a more coarse-grained simulator whose goal is to be fast to simulate a multi-year workload of multiple large resources. This is important for the rapid development of user models and quantifying parameterization uncertainties. Later, the developed user model can be applied to the Slurm simulator.

The key concept in this work is the user's compute need representation. The user research projects can be represented as multiple possibly dependent compute tasks (see Figure 1-B for illustration). Each task can represent multiple simulations with different initial conditions for statistical purposes or a system of interest vs control. Each task is qualified by the number of node hours it requires to be completed. It is up to the user how to execute the task, that is, split it into batch jobs. The user can split a job in multiple ways. For example, a ten-node-hour job can be done on one node in ten hours, ten nodes in one hour, or two consecutive jobs on one node for four hours followed by two nodes for three hours (here, we assume the task can be scaled perfectly). The user can make a decision based on past experience, colleague recommendations, and currently available resources.

The compute task approach was inspired by the author's experience in Molecular Dynamics (MD) simulation of biomolecular systems (Simakov et al., 2017; Simakov and Kurnikova, 2010; Volynsky et al., 2005). MD is widely used to simulate biological macromolecules like proteins, RNAs and DNAs (for reviews, see (Hollingsworth and Dror, 2018; Karplus and Petsko, 1990). The MD simulation is one of the largest cycle consumers on HPC resources. In ACCESS HPC resources, 5 to 15% of all cycles were consumed by MD applications (as reported by XDMoD). In MD, all atoms are represented as classical particles, and the second Newton equations are solved to calculate atomic movements. During macromolecules simulation, it is common to have a single simulation that requires days or weeks of computation, while HPC resources typically enforce a one—to three-wall time limit (for MD reviews, see). Thus, users must split the overall simulation into a series of smaller jobs. One characteristic of such a job is running all the way till the requested wall time. On TACC-Stampede2 (Intel Skylake-X partition), the fraction of MD jobs exceeding the requested wall time is 44% (by total core hours obtained from ACCESS XDMoD), which is similar to 43% in all other applications. Thus, it is highly likely that a large portion of MD jobs are resulting from splitting larger compute tasks into a series of jobs. Furthermore, a similar process

is observed in other applications, and thus, the compute task concept can be generalized to other fields of science. Figure 1.A illustrates a typical workflow for a multiple-group MD simulation study, and Figure 1.B shows its computational task workflow representation. Other MD workflows, like Free Energy Perturbation (FEP) and alchemical transformations, can also be represented by similar computational task workflows. In this work we concentrate on a specific application type and model a particular workflow within this field which allow future model due to better understanding of the domain. A good understanding of domain would allow to have more accurate model and it can be later generalized to other fields of science.

This work introduces and implements a compute task workflow representation of users' computational projects. The task splitting into compute jobs is a function of users' computing needs/characteristics, available hardware, and imposed resource limits. We have implemented three strategies: 1) historical replay - every compute job is its own task (user for scheduler verification), 2) preferred task split – the reconstructed compute tasks are split based on user's preferred size (obtained from historic jobs of the user) and 3) adaptive task splitting – user can modify the job's nodes and wall time request to fit into available resource slot. The strategies were tested on a tiny ten-node system as well as a large, almost two-thousand-node machine.

RELATED WORK

The analysis and modeling of the HPC workload are important for understanding users' needs and parts of the system (for example, the job scheduler), as well as future systems design. Therefore, it is understandable that it was a subject of study for a long time. One of the oldest related studies describes the probability call distribution in telephone queues in the early operator era of telephony (Erlang, 1909). The most comprehensive review on HPC workload analysis and modeling is a book by Dror Feitelson (Feitelson, 2015), specifically chapter 8 for modeling. Most workload modeling approaches concentrate on users' activity cycles (working hours and days) and are job-centric and application-agnostic. For example, jobs are drawn from distributions parameterized from historic workload (Cirne and Berman, 2001a; Jann et al., 1997; Lublin and Feitelson, 2003). Another approach is to resample jobs from historic workload (Zakay and Feitelson, 2013). Finally, historical workloads can be used directly. These approaches do not allow for job size change once it is drawn or taken from the distribution. This complicates the modeling of the feature system. Number of works are addressing it by allowing job size to change (Cirne and Berman, 2001b; Downey, 1998). However, it is still job-centric and assumes that the jobs are self-contained and independent. Zhou and others (2024) use autoregressive regression to add dependencies between jobs.

Most of the abovementioned work is batch job-centric, and in many cases, the user is not modeled explicitly. In this work, we treat the batch

jobs as a manifestation of the user's computing needs, the performance of the hardware, and the limits imposed by resource providers (max node counts, max wall time, and so on). The user can take dynamic action on the resources available. For example, they can reduce their calculation needs or leave the system if the system is too busy, on the over side if the system is empty, they can use larger job sizes (Alvarez et al., 2016) and increase the compute needs by using more advanced methods or improved statistics. Omitting user behavior can be a significant problem for HPC resource modeling. Another important aspect is that the job from the same task is automatically dependent: their size and wall time are similar or correlated. The job correlation is often referred to as self-similarity (Feitelson, 2015) and many job-generating approaches require separate handling.

METHODS

HPC resources typically utilize a batch jobs processing scheme, where the users submit their calculations and resource request specifications (typically written in a script file and called a batch job) to the HPC resource workload manager (often called a scheduler). Then, the scheduler allocates the requested resource and starts the calculations; if the resources are unavailable, the scheduler postpones the batch job execution until resources are available. The jobs that cannot be executed right away will pile up in the queue, and it is up to the scheduler to determine which job will be executed first.

The developed framework is implemented in Julia language and utilizes Agent.js framework (Datseris et al., 2022) for Agent-Based Modeling. The first-in-first out (FIFO, also called first-come-first-served, FCFS) scheduler with simple backfilling was implemented. The backfill schedule schedules the jobs, which does not affect the start time of the next job to be executed by the FIFO scheduler. The implemented ABM utilizes a discrete stepping algorithm and uses a one-hour step by default.

User's workflow representation and compute tasks. The users' computing needs are represented as a collection of compute tasks described as a certain number of node hours, and the user converts each compute project to a series of compute jobs and submits them sequentially (the next job is submitted after the previous one is completed) to the HPC resource schedule. The HPC scheduler places jobs on the available compute node; if no resources are available, the job is placed into the waiting queue until the resource becomes available. For example, a user's computing project requires 32 node hours; it can be executed in multiple ways: a single 32-hour job on one node, two sequential 16-hour jobs on one node, one 16-hour job on two nodes, and so on. It is up to the user how to split the compute project; the user's decision can be based on previous experience, other users (from the same research group), or currently available resources.

Compute tasks splitting strategies. In this work, we implemented three schemas for computing task splitting into batch jobs. 1) In replay mode, the user submits previously described jobs. This mode is good for replaying historical workload, and here essentially, each job is its own task. 2) In the

user's preferred mode, the job size and duration are preset for a task and represent what the user thinks is the appropriation size for a given problem and HPC resource size (here, the user does not consider current resource utilization). 3) In the adaptive mode, the user adjusts the node size within the 0.5 to 2.0 range of preferred value and wall time within the 0.25 to 4.0 range of preferred value based on currently available resources. The preferred node count and wall time were set to their quantile of jobs within the same task. That results in overall closest wait time.

Parallel scaling. In the MD, different simulated systems will scale differently on different hardware. In this work, we abstract from the scaling problem and assume perfect scaling within a range of the user's preferred value (obtained from the historic workload). This is reasonable as we only allow the job size to change from half of the preferred setting to double the preferred size.

Parameterization of thinking time. Thinking time is the time between the end of the first job and the submission of the next job. The user needs this time to realize that the job is completed and to write the next job script. It also would include out-of-office time. For parameterization, TACC-Stampede2 historical workload was used. If there were multiple concurrent jobs in the queue for a given user the thinking time was calculated between the last end time of that job and the first next one. The thinking time distribution was fit to gamma distribution using moment matching estimation and R fitdistrplus library (Delignette-Muller and Dutang, 2015).

Compute task restoration from batch jobs. The compute task was identified from the historic workload by combining jobs with the same node count. The finished job is matched to the closest next one if it was submitted after the first one, had the same node count, was submitted within one week, and was not picked by a different compute task.

Table 1. A test four compute-tasks workload for a ten-node cluster.

Task Id	User Id	Submit Time, Hours From Start	Compute-Required, Node-Hours	Preferred Number of Nodes	Preferred Requested Walltime, Hours
1	1	1	140	4	48
2	2	2	140	4	48
3	3	3	140	4	48
4	4	4	140	4	48

Tiny, test, system. A small 10-node cluster with 4-compute tasks (see Table 1 for task description) was developed for testing purposes and to illustrate capabilities. It is also used for regression tests within the framework of continuous integration (CI) tests.

Large system. An Intel Skylake-X portion of the TACC-Stampede2 system. It consists of 1,736 nodes. The data was obtained from the ACCESS instance of XDMoD (Palmer et al., 2015). The five-year workload was rounded to the nearest hour and included jobs that were at least one hour long. The resulting workload consists of 967,388 jobs. For the task-splitting study, only users utilizing Gromacs (Abraham et al., 2015) and NAMD (Phillips et al., 2020) Applications (MD application) were used. The 36 selected MD users (1.3% out of the total 2,600 users) are responsible for 4.1 % of all cycles consumed.

Hardware. All calculations were done on a laptop with an Intel Core i7-11800H processor and 64GiB memory.

Source code, examples, and analysis scripts are available at:
https://github.com/ubccr-slurm-simulator/HPCMod.jl
https://github.com/ubccr-slurm-simulator/HPCMod_Article_AHFE24

RESULTS AND DISCUSSIONS

Small 10-Node Cluster with 4-Compute Tasks. For testing purposes and capabilities illustration, we used a small 10-node cluster with 4-Compute Tasks (see Table 1 for task description). These four tasks would produce four compute jobs with the default preferred node count and wall time (Figure 1.A). Using compute tasks to describe the user's computational needs allows us to model different aspects of user interaction with an HPC resource. For example, the system administrators might reduce the maximum allowed wall time to 24 hours from 48 hours to improve server serviceability. This results in a doubling of jobs and an increase in time-to-solution (time from submit time to the completion time) for half of the tasks (Figure 1.B). With computing tasks and a user-based modeling approach, we can also model different user behaviors on the system. For example, user 4 (responsible for task 4) can try to take advantage of the backfill scheduler and modify the requested nodes and wall time based on the available spot till the next job in the queue. In our toy test that resulted in having two jobs for task 4 (Figure 1.C). Task 4 finished faster overall, but the user needed two smaller consecutive jobs instead of one big one. In this simulation, the second job was submitted right after the completion of the first one. The actual user would need some time after the first job was completed and before submitting the second job. This time is often called think time and can include factors like user working hours and job check frequency (often referred to as job babysitting by graduate students). The results of the simulation with a more realistic think time drawn from random distribution parametrized from historic jobs are shown in Figure 1.D.

Certainly, we can not draw any conclusions from this toy system other than that HPCMod can make such simulations, and we need to examine more realistic systems further.

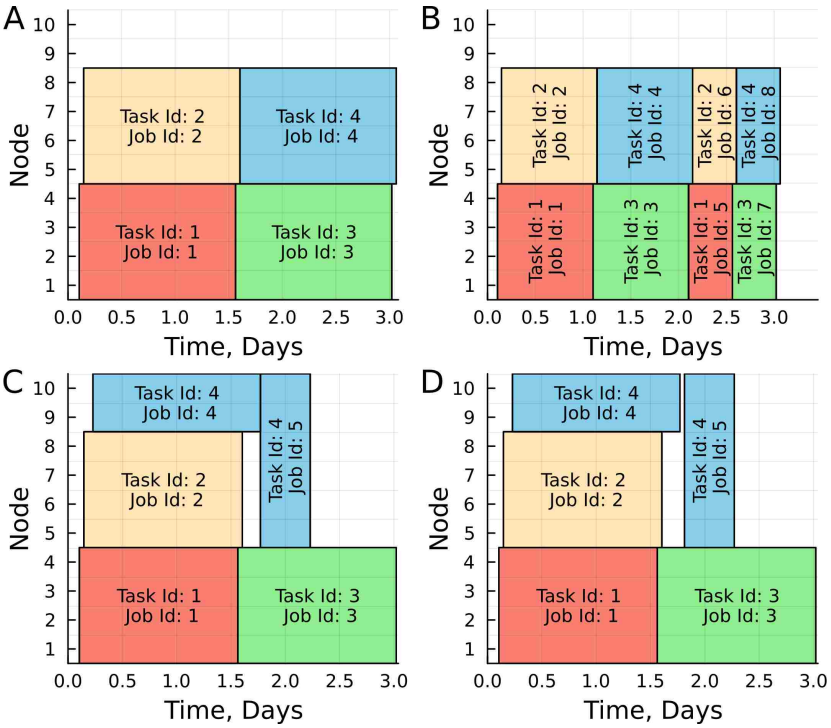

Figure 2: A test workload of four compute tasks (see Table 1) on a ten-node cluster. Colors highlight the task occupation, and black lines outline the job borders. **A**. All users utilized the preferred node number and wall time for job requests. **B**. Same as A, but the max allowed wall time was reduced to 24 hours. **C**. Similar to A, the fourth user uses an adaptive strategy to select node count and wall time. **D**. Similar to C, the fourth user uses an adaptive job specification strategy, and gamma distribution for think time is used; note the gap between jobs 4 and 5.

Simulation of TACC Stampede-2 SKX (1,736 Node System). For a large-scale test, we simulated a 5-year-long workload of large 1,736 node machines. To validate the reasonability of our scheduler, we first compare the historical workload replay to the actual historical realization (Table 2). The simulator produces a total wait time 8% higher than a historical realization. During the simulation, only jobs longer than one hour were used, and a FIFO scheduler instead of a priority-based scheduler was used. Therefore an 8% difference is qualitatively reasonable.

Next, to validate our compute task restoration algorithm, we conducted a base simulation in which the compute needs of selected MD users (36 out of 2,600 users) were represented as compute tasks. The compute jobs were generated from these tasks using historic node count and the third quantile of historic wall times (the third quantile produces the best outcome; maximum, mean, medium, and first quantile were also tested). The wait time for compute task users is 10% lower than in historical replay, which provides reasonable qualitative agreement. This simulation is called "base" because that is a starting point to compare different strategies.

In the adaptive simulation, the selected MD users were allowed to modify their job requests based on the HPC resource occupation. The users were

provided with currently available nodes and time availability until the first job in the queue used them. Given this information, the user can modify the requested node counts (from half to double the preferred value) and wall time (from quarter to quadruple the preferred value). Interestingly unlike in our synthetic 10-node system, the number of jobs decreased almost twice for the compute task users. The total wait time also decreased almost twice (45% decrease). However, this comes with a wait time increase of 4% for other users.

A single simulation of the 5-year-long workload takes 193 seconds to complete, which is 800,000 times faster than real-time and much faster than ~500 times of the current version of UB Slurm Simulator (Simakov, 2024). Due to stochastic processes in the HPCMod (currently only think time), running the same simulation multiple times is important. In the future, adding more random processes for addressing uncertainties in the parameterization would further increase the need for multiple runs. That is another area where a Julia language built-in multi-threaded and distributed computing capability will be useful. In the future, we anticipate that a fast coarse-grain simulation will be done with HPCMod, followed by a more detailed study will be done with the Slurm Simulator.

Table 2. Results of Stampede 2-SKX simulation. MD users are users for which a compute task schema was used (36 MD users).

Simulation	Total Wait Time, Hours	Jobs Number for Static Users	Jobs Number for MD Users	Total Wait Time for Static Users, Hours	Total Wait Time for MD Users, Hours
Historical realization	17,554,045	955,025	12,362	16,941,928	612,117
History Replay	18,899,615	955,025	12,362	18,675,684	223,931
Base	19,176,135	955,025	12,946	18,975,528	200,607
Adaptive	19,755,326	955,025	6,576	19,643,311	112,015

CONCLUSION

We introduced a compute task workflow specification of the users' computational project and implemented it within HPCMod, an HPC resource modeling framework. The framework was tested on a small ten-node cluster and a large 1,736-node system modeled after a portion of the TACC-Stampede-2 system. A simple algorithm was used to reconstruct the computing task from the historical workflow of MD users. This allows the testing of different task-splitting techniques. The historical replay was used for the job scheduler testing. This method was also used for most of the users during the selected user's alternative task-splitting testing. The selected users (large MD users) were allowed to use preferred task splitting; this validated the task reconstruction algorithm. Finally, the selected MD users utilized an adaptive task-splitting strategy, where the user can change job size and requested wall time to improve the odds of being

scheduled by backfiller. In the toy cluster, such a strategy decreased the time to a solution but increased the number of jobs. The strategy was extremely successful in large systems, decreasing the wait time by 50%. However, all other users got a 4 % increase in their wait time. This conclusion should be taken with a grain of salt as it is done only for one realistic system and only for a small selection of users, and further studies are needed. The implementation is very fast, and we anticipate that with further development of the user model, that will decrease a bit. We plan to improve the user model further and incorporate the quantification of parameterization uncertainties.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation under the OAC 2004954 and 2137603 awards.

REFERENCES

Abraham, M. J., Murtola, T., Schulz, R., Páll, S., Smith, J. C., Hess, B., Lindah, E., 2015. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX 1–2, 19–25.

Alvarez, G. P. R., Ostberg, P. O., Elmroth, E., Antypas, K., Gerber, R., Ramakrishnan, L., 2016. Towards Understanding Job Heterogeneity in HPC: A NERSC Case Study. Proceedings - 2016 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2016 521–526.

Cirne, W., Berman, F., 2001a. A comprehensive model of the supercomputer workload. 2001 IEEE International Workshop on Workload Characterization, WWC 2001 140–148.

Cirne, W., Berman, F., 2001b. A model for moldable supercomputer jobs. Proceedings - 15th International Parallel and Distributed Processing Symposium, IPDPS 2001.

Datseris, G., Vahdati, A. R., DuBois, T. C., 2022. Agents.jl: a performant and feature-full agent-based modeling software of minimal code complexity. Simulation.

Delignette-Muller, M. L., Dutang, C., 2015. fitdistrplus: An R Package for Fitting Distributions. J Stat Softw 64, 1–34.

Downey, A. B., 1998. A parallel workload model and its implications for processor allocation. Cluster Computing 1998 1:1 1, 133–145.

Erlang, A. K., 1909. The theory of probabilities and telephone conversations. Nyt Tidsskrift for Matematik B 20, 33–39.

Feitelson, D. G., 2015. Workload modeling for computer systems performance evaluation, Workload Modeling for Computer Systems Performance Evaluation. Cambridge University Press.

Hollingsworth, S. A., Dror, R. O., 2018. Molecular Dynamics Simulation for All. Neuron 99, 1129–1143.

Jann, J., Pattnaik, P., Franke, H., Skovira, J., Riordan, J., 1997. Modeling of workload in MPPs. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 1291, 95–116.

Karplus, M., Petsko, G. A., 1990. Molecular dynamics simulations in biology. Nature 1990 347:6294 347, 631–639.

Lublin, U., Feitelson, D. G., 2003. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. J Parallel Distrib Comput 63, 1105–1122.

Palmer, J. T., Gallo, S. M., Furlani, T. R., Jones, M. D., De Leon, R. L., White, J. P., Simakov, N., Patra, A. K., Sperhac, J., Yearke, T., Rathsam, R., Innus, M., Cornelius, C. D., Browne, J. C., Barth, W. L., Evans, R. T., 2015. Open XDMoD: A tool for the comprehensive management of high-performance computing resources. Comput Sci Eng 17, 52–62.

Phillips, J. C., Hardy, D. J., Maia, J. D. C., Stone, J. E., Ribeiro, J. V., Bernardi, R. C., Buch, R., Fiorin, G., Hénin, J., Jiang, W., McGreevy, R., Melo, M. C. R., Radak, B. K., Skeel, R. D., Singharoy, A., Wang, Y., Roux, B., Aksimentiev, A., Luthey-Schulten, Z., Kalé, L. V., Schulten, K., Chipot, C., Tajkhorshid, E., 2020. Scalable molecular dynamics on CPU and GPU architectures with NAMD. J Chem Phys 153.

Simakov, N. A., 2024. Slurm Simulator Development: Balancing Speed, Accuracy, and Maintainability (Poster). International Conference on High Performance Computing in Asia-Pacific Region (HPCAsia 2024), January 25–27, 2024, Nagoya, Japan.

Simakov, N. A., Deleon, R. L., 2024. (Poster) Slurm Simulator Development: Balancing Speed, Accuracy, and Maintainability. In: International Conference on High Performance Computing in Asia-Pacific Region (HPCAsia 2024), Nagoya, Japan.

Simakov, N. A., Innus, M. D., Jones, M. D., DeLeon, R. L., White, J. P., Gallo, S. M., Patra, A. K., Furlani, T. R., 2018a. A slurm simulator: Implementation and parametric analysis. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 10724 LNCS, 197–217.

Simakov, N. A., Jones, M. D., DeLeon, R. L., White, J. P., Innus, M. D., Gallo, S. M., Patra, A. K., Furlani, T. R., 2018b. Slurm simulator: Improving Slurm scheduler performance on large HPC systems by utilization of multiple controllers and node sharing. ACM International Conference Proceeding Series.

Simakov, N. A., Kurnikova, M. G., 2010. Soft wall ion channel in continuum representation with application to modeling ion currents in α-hemolysin. Journal of Physical Chemistry B 114, 15180–15190.

Simakov, N. A., Leonard, D. A., Smith, J. C., Wymore, T., Szarecka, A., 2017. A Distal Disulfide Bridge in OXA-1 β-Lactamase Stabilizes the Catalytic Center and Alters the Dynamics of the Specificity Determining Ω Loop. Journal of Physical Chemistry B 121, 3285–3296.

Volynsky, P. E., Polyansky, A. A., Simakov, N. A., Arseniev, A. S., Efremov, R. G., 2005. Effect of lipid composition on the "membrane response" induced by a fusion peptide. Biochemistry 44, 14626–14637.

Yoo, A. B., Jette, M. A., Grondona, M., 2003. SLURM: Simple Linux Utility for Resource Management. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 2862, 44–60.

Zakay, N., Feitelson, D. G., 2013. Workload resampling for performance evaluation of parallel job schedulers. ICPE 2013 - Proceedings of the 2013 ACM/SPEC International Conference on Performance Engineering 149–159.

Zhou, Z., Sun, J., Sun, G., 2024. Automated HPC Workload Generation Combining Statistical Modeling and Autoregressive Analysis. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 14521 LNCS, 153–170.