A Model for Performance Evaluation

[1]Amir Antonie and [2]Andrew Mathus
[1,2] School of Computer Science, University of Guelph, Guelph, ON N1G 2W1, Canada.
[1]antonieamirjohn@hotmail.com

Article Info
Journal of Computing and Natural Science (http://anapub.co.ke/journals/jcns/jcns.html)
Doi: https://doi.org/10.53759/181X/JCNS202202001
Received 10 May 2021; Revised form 15 June 2021; Accepted 20 September 2021.
Available online 05 January 2022.

Abstract - As a result of the parallel element setting, performance assessment and model construction are constrained. Component functions should be observable without direct connections to programming language, for example. As a result of this, solutions that are constituted interactively at program execution necessitate recyclable performance-monitoring interactions. As a result of these restrictions, a quasi, coarse-grained Performance Evaluation (PE) approach is described in this paper. A performance framework for the application system can be polymerized from these data. To validate the evaluation and model construction techniques included in the validation framework, simplistic elements with well-known optimization models are employed.

Keywords - Performance Evaluation (PE), High Performance Computing (HPC), Common Component Architecture (CCA).

I. INTRODUCTION

To mitigate data-intensive issues, scientific technology is usually performed in high-performance ecosystems. Performance of the application is largely determined by two factors in this setting: the (effectual) processor velocities of the independent CPUs and the communication duration between processing units. Many times, enhancing data redundancy will improve the performance of an existing CPU. When using shared memory machines (MPPs and SMP hubs), inter-processor interaction is often attained through data transmission, which has a significant impact on the application's parallelization and load-balancing. To reduce the production cost, algorithmic techniques are used, such as interchanging information exchange with arithmetic operations and reducing global substantial reduction and impediments.

Timers and device countertops are the usual tools for measuring performance. To determine the speed at which code executes, a user uses a timing device (or a thermostat). As an example, hardware registers keep track of cache misses. For the application interface, both of these methods focus on providing insight into what happens when the program is run on a particular machine. To measure the network throughput in a parallel setting, publicly released techniques track performance measures such as the amount of data exchanged between processing units and the amount of time expended sending messages between the processing units. It is possible to create a conceptual framework for the application based on measurements data. This prototype shows the expected performance for a given collection of issue and console parameters. If you are transferring an existing coding standard to architecture, you will need to conduct Performance Evaluation (PE) and model construction as part of the design and optimization stage. For example, performance frameworks could be used to estimate throughput.

Recently, the research establishment has been pushing for the definition of a component-based software package for High Performance Computing (HPC) systems [1]. An increasing diversity of science-based simulation code as well as a desire to expand multidisciplinary approach is driving this change. After this interest grew a team of researchers connected with diverse educational organizations and government laboratories formed the Common Component Architecture (CCA) Discussion board. The HPC constituent framework was suggested by the CCA Discussion board. There are a couple of unique problems when it comes to measuring performance and model construction in component-based applications. An application's performance is influenced by the efficiency of the independent units as well as their interplay with each other.

Furthermore, a given element may have numerous practical applications (i.e., elements, which provide similar functionality but apply varied algorithms or utilize varied internal structures of data). If you choose deployments at random, you may end up with a component assembly that is technically correct but not optimally configured. A constituent landscape must therefore be capable of measuring and designing performance in order to analyze performance of an element in the setting of a combined system arrangement. Direct instrumentation of the code in component-based surroundings might not even be possible, in compared to conventional homogeneous standards. In many cases, the operator of an element is not the element's creator, and would not have direct exposure to the element's code. A quasi coarse-grained measurement framework which can evaluate software at the constituent is needed to address these constraints.

By using proxy servers to instrument a high - performing element application, a quasi, coarse-grained measurement model is described in this article. In order to produce performance frameworks for independent units, the measurement data can be used to create an overall software conceptual framework. As an added bonus, this paper portrays how to ascertain an estimated optimal element configuration for finding an optimal solution. The rest of this research is arranged as follows:

Section II reviews the past literature works. Section III critically analyses the theme of the paper. Finally, Section IV concludes the paper and presents future research directions.

II. LITERATURE REVIEW

There are a number of programs related to the research topic. Those projects are described in this section, along with their significance to the works at hand.

N. Parlavantzas and G. Coulson in [2] conducted an analysis of the software component frameworks. According to the study, the three most common resource software component frameworks are CORBA, COM and DCOM, and Java Enterprise Beans. Such frameworks are insufficient for high-performance computer technology since they do not guarantee effective parallel correspondence, appropriate data abstract concepts (e.g., intricate metrics or multi-dimensional clusters), and/or will not provide query languages. Due to the fact that these conventional frameworks are designed for sequential application areas, they have minimal reasons for concerns with the hardware details and hierarchy of memory of the prevailing physical facets that are crucial in Cloud computing.

M. Aldinucci, V. Cardellini, G. Mencagli and M. Torquati [3] presented a difference between platforms of non-HPC elements frameworks are directed to. As a result, many of their software system are accomplished over wide area networks (WANs) or local area networks (LANs); HPC implementations have been almost primarily run on intricately intertwined hubs of MPPs (Massively Parallel Processors) or SMPs (Scalable Memory Processors). Since these conventional architectures have strong reliability communications round-trip intervals and signal timeouts, they are inappropriate for use in high-performance computing.

S. Kaliraj and A. Bharathi in [4] conducted an analysis of the component measurement frameworks. Despite all the problems of conventional systems, these frameworks often use similar methods when evaluating performance. This article describes a proxy-based framework for the Java Beans conceptual framework. Proxy servers are set up for each surveyed element, intercepting calls and sending them to a central monitoring element. Displaying data is the sensor's responsibility as well as sending out messages. There is a primary goal of this conceptual model: to pinpoint hotspots or elements which do not balance well in the constituent application's modules. In order to evaluate the performance of the CORBA conceptual framework, the WABASH instrument is used. This instrument is designed primarily for measurement and analysis of decentralized CORBA systems prior to deployment. "Interceptors" is the name given to a WABASH indicator that handles input and output queries for all amenities (servers). In addition, WABASH characterizes an executive portion that is fully accountable for checking the interceptors in order to retrieve data and to implement plans.

Imperial College London's Parallel Software Group uses grid-based constituent computer technology. Their proposition for performance management is also centered on proxy-based data collection and analysis methods. On the basis of performance modeling techniques and resource availability, each component's execution will be optimized [5]. This means that there are a total of n installations to choose from. Performance models and performance requirements of an application must be submitted by constituent developers into a repository by the constituent developer's deadline. In order to replicate the software and create a call-path, proxy servers are used. To generate a polymeric measurement system, examine each command “ in the call-path generated by proxy servers. Parameters are used instead of integration references to guarantee that the composite prototype is distinct of integration. Instead of parameters, integration performance frameworks can be used to approximate operational costs. This is done by selecting the model that returns the lowest cost, and then creating an operational plan based on that prototype.

T. LIU, B. FAN, C. WU and Z. ZHANG in [6] conducted an analysis on runtime optimization. It is worth noting that Autopilot and Active Harmony are two activities with similar techniques to executable maximization of code. Surveillance and optimization infrastructures are required for both system applications. All these variables are disconcerted during implementation in Active Harmony, and the results are tabulated. It utilizes a symmetric automated system to determine the optimal model parameters while Autopilot utilizes inductive reasoning to determine the optimal set of variables. When it comes to either system, it is vital to decrease the number of variables by recognising which "negative" portions are avoided by the framework. Libraries can be substituted using Active Harmony's architecture, allowing for more optimum solution integration.

B. Palmer, Y. Fang, V. Gurumoorthi, G. Hammond, J. Fort and T. Scheibe in [7] evaluated the CCA performance monitoring and measurement. When it comes to optimizing the CCA architectural style and schema, a self-managing element enhanced version to the CCA systems has been proposed. These adjustments involve adjusting elements to endorse measurement and management functionality, as well as broadening the framework that supports rule-based management measures, among other things. There are two extra forms of elements in the network: an element supervisor and a structure supervisor. The element managers are in charge of intra-component judgements, while the structure supervisor is in charge of inter-component actions and decisions. There are high-level laws that govern the interactions and behaviors of elements.

As a result, Argonne National Laboratory's ongoing study has a really similar ethos. Architecture for performance analysis, database administration and responsive combinatorial optimization for parallel element PDE simulation models is proposed in the research [8]. PDE-based simulation models rely heavily on nonlinear and sequential computational modules to carry out the bulk of the arithmetic operations. Therefore, these are the two aspects that this software construction supervises. Measured values from TAU-instrumented code are stashed in the PerfDMF dataset (comparable to

the work reported in this article). As a result of the architecture, it is hoped that the test data and modeling techniques will be used to assist in generating multi-method solutions for managing massive linear equations systems of linear equations.

This study also tries to integrate a quality of service component into its findings. Persistent databases are used to store software characteristic features that have an influence on productivity such as convergence speed, parallel scaling, and precision in the form of metadata. They can be assessed including the performance whenever a decision is being made, as well as the other attributes.

III. CRITICAL ANALYSIS

The Common Component Architecture (CCA)

The Common Component Architecture (CCA) is a range of principles that defines a lightweight components particularly optimized for maximum computation power. To date, emerging resource component software frameworks, like Microsoft's COM and DCOM and Enterprise Java Bean, have been deemed insufficient for HPC for different reasons [9]. Because of this, they were ill-equipped to effectively manage efficient parallel information exchange, did not provide requisite scientific evidence abstractions (e.g., intricate statistics and multifaceted configurations), and/or had limited language interconnectivity. In the CCA prototype, components to communicate through well-defined, interoperability, allowing them to work together seamlessly. By modifying one constituent or substituting another element that integrates the same functionality, it is possible to modify applications. Plug-and-play is among the main reasons why software application frameworks exist today.

The CCA configuration is centered on the communication system of Provides-Uses (P-U). Functionality is provided by elements through the integrations they extract. Other elements offer features and functions through their integrations. Terminals are the names given to the integrations. It loads and instantiates a combination of factors in the schema. In fact, the CCA does not provide a structure at all, but rather a configuration for one. Due to this, there are a number of CCA-compliant architectures on the market today. A framework called Ccaffeine has been used in the investigation described in this study at Sandia National Laboratory facilities. SCMD (Single Component Multiple Data) symmetric communication system is used in Ccaffeine. It is indeed basically SPMD (Single Program Multiple Data) parallel processing inference applied to constituent software. To put it another way, the constituent functionality is reproduced on every processing unit, and each central processing unit enacts the software.

CCA software is shown in Figure 1. Elements that have been instantiated can be seen in the Arena section. A module's supplier terminals are on the left, while the utilised terminals are on the right. They would be the same form in able to link a P-U port to another P-U port. A disconnected Go port is provided by the driver constituent. By using the conceptual model, the port's configuration can be changed.

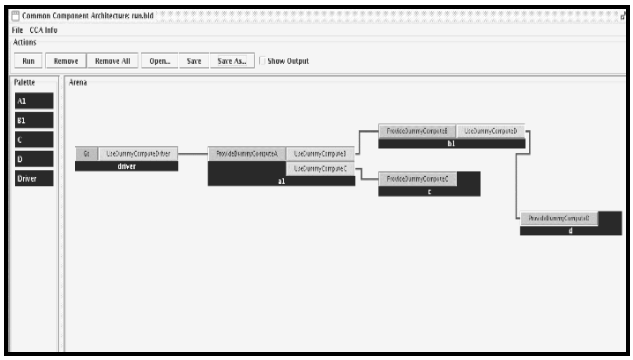

Fig 1. Diagram for simplistic CCA software wiring

Performance Measurement Model

This means that the polymeric efficiency of a constituent application is based on the efficiency of each constituent and the effectiveness of their interrelationships. With no consideration for how an element will be used, its efficiency will be inconsequential. Three variables define this framework: (1) the issue being rectified, (2) the insight to an element, and (3) the interplay between the person on the phone and callee modules. Imagine an element that is needed to perform two major functions on massive quantities of data in parallel. Both functions use strided direct exposure. Evidently, the performance of the constituent is dependent on the problem under consideration and the configuration of significant exposure being used in order to fix the issue. Additionally, the module's insight affects its performance results. For both configurations of connectivity in the preceding example, the efficiency of the arrangement will vary in magnitude. If an element has to do more work, the runtime will be longer.

Lastly, there is the possibility of misaligned types of data between constituents. Presumptions and types of data are shared across the entire program in homogeneous code, resulting in a single application. In a component-based ecosystem, this is not true. Data transformations must be considered if they are required for communication between two modules. Constituent configurations can be optimized once an element context has been characterized. Most scientific modules are executed in phases that require a lot of computing power after which there are calls for inter-processor information

exchange. Some of these calls could be used to communicate. These include interoperability expenditures, such as a global minimisation or barrier. As a result of the generalizing presumptions in this research, the renovation of the optimization models has been made more straightforward. This means that all interaction between processors is presumed to be impeding.

Therefore, no overlaps are reported between computation and communication. This establishes a clear variation between the communication stages and computation phases. As a result, correspondence and information processing do not overlap. In this way, a clear difference is drawn between communication and computation. Therefore, in order to build up a component's performance measure (for each initialization of a technique extracted via the Provides Port), the following must be considered:

- The time of execution
- The amount of time spent during inter-processing communications – this could be illustrated by mathematical expressions of the overall inclusive time taken in the various MPI calls which have their origins from the invocation or initialization techniques.
- The timeframe of computation that is determined as the variation between two separate values.
- Input parameters, which impact the overall performance. Typically these parameters will incorporate data sizes being worked on (such as the array sizes) or/and measures of particular repetitive operations (such as the times a smoother might be used in the multi-grid application).

The "Tuning and Analysis Utilities" (TAU) Toolkit, for example, makes it easy to meet the first three criteria. The fourth requirement calls for a comprehension of the automated system which a distinctive constituent integrates in order to complete the project. In addition, constituent surroundings create major challenges in performance management. Because the application developer would not have direct exposure to a module's code, a quasi-measurement framework is necessary. As a second consideration, the developer is focused as to how an element behaves in a situational setting. Optimizing software is more important than enhancing a component's performance from the software developer's perspective. Daher is a method-level qualitative technique of an element important to the application programmer. A proxy-based measurement framework, relevant to the research work, offers an effective architecture that meets both prerequisites. A detailed description of this proxy-based system can be found in the section below.

In order to evaluate, collate, and document the performance measurements for a constituent application in the measurement tool, 3 separate constituent types must be used.

The types include:

- The collection of proxy constituents
- Measurement constituents
- Manager constituents

Proxy Constituents

A proxy constituent is a lightweight constituent that is rationally positioned at the front of the constituent being evaluated. Thus every technique interrupt to the embedded system is intercepted and forwarded by the proxy. Caging the callback function allows the user to monitor its success. It is important to note that in a CCA surrounding, the proxy would provide (and use) relatively similar ports as the element to be configured. To see how a constituent and proxy are related, see Fig 2. Example: Constituent C has two provisioning ports, "a" and "b" positioned on its left side, as shown in this image. "Uses" ports are situated on the right hand side of the constituents and are differentiated using ("") apostrophe.

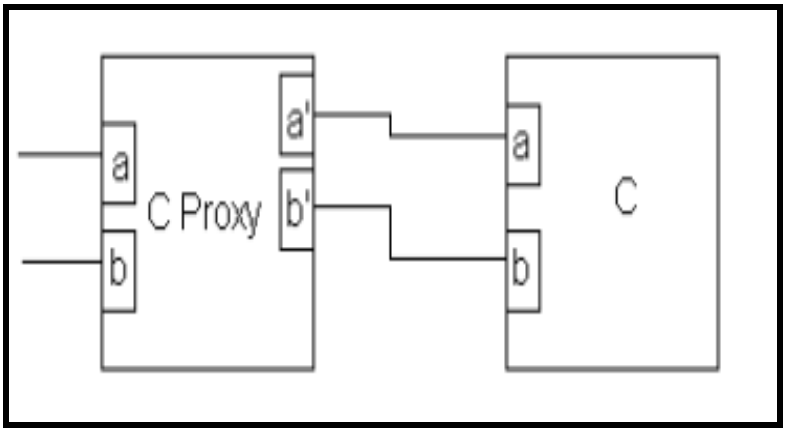

Fig 2. Constituent instrumented using one proxy integrated "in front", which forwards and intercepts all technique calls to the constituent C.

Proxy creation is automated using a tool that takes advantage of the intrinsic features and dependable development of the proxy [10]. C++ open source evaluation and computation is enabled by the Software Database Toolkit, which the

instrument is built upon. Initial proxy creation was done constituent by constituent. To supervise all the terminals of a constituent, a proxy had to be developed by hand. As opposed to this method, which generates a proxy for only one port, the proxy system does not. Creating multiple proxy servers for a constituent with different ports is a common practice today. Figure 3 below represents the configurations in case the proxy system is utilized to tool constituent C from before.

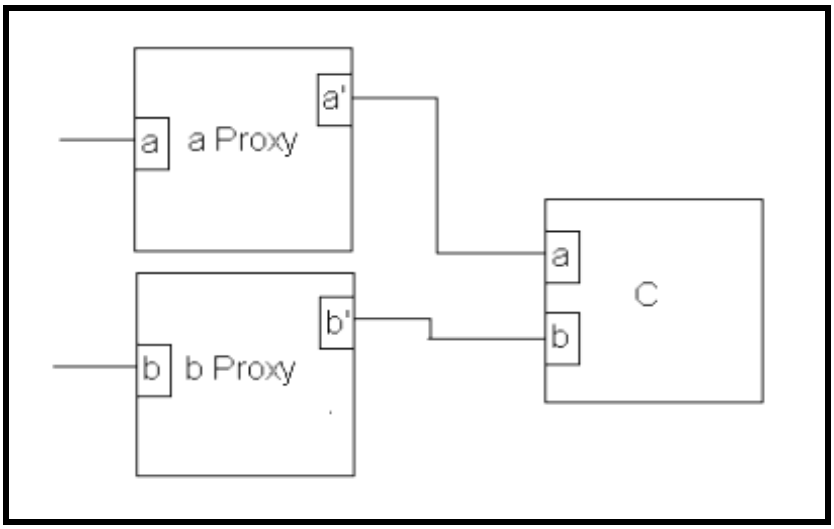

Fig 3. Constituent instrumented by wide-range proxies (a single proxy in every "provides" port)

Measurement Constituent

When it comes to timer control and implementation, and also interacting with the operating system of the computer, timers are the second category of measuring device infrastructures. Using the TAU constituent, which acts as a wrapper for the TAU measuring device library, this is accomplished. When it comes to measuring efficiency, TAU offers both categorizing and documenting alternatives. With this constituent, you can create a measurement port for signalling and management services. You can also track recollection and perform evaluation queries. Using the synchronization functionality, you can create timers and name them as "begin", "quit", and "band" them.

As a result of its event functionality, the constituent can keep records of application and sourced atomic occurrences. We document the upper and lower limits from each occurrence as well as the variance as well as the relative frequency. The Performance Application Program Interface (API) and the Efficiency Counter Library are two external libraries that TAU uses to communicate with computation device counters and high-performance loops. Clusters of trackers can be activated using the countdown control system. TAU, for instance, can be constructed to instrument MPI schedules through the MPI Profiling Functionality. One call could activate or deactivate all of the MPI schedules at once. Programs using the database management system are able to get their hands on all the performance data that is collected. It also dumps description performance profiles upon software cessation, thanks to the TAU Library.

Manager Constituent

Mastermind's Manager in charge of capturing, storage, and disclosing performance information. WatchPort is a Mastermind feature that allows proxy servers to begin and halt surveillance. When a function is supervised, a file is created for every technique which is used by the element in question. Each time the technique is invoked; this documentation is modified with the performance information for the technique in question. We keep track of the input variables and communication and computation phases of every invocation. Every record discharges its performance information to disk upon software dismissal. The integration of multiple types of components allows for quasi instrumentation of a constituent solution. If the software is configured, the Mastermind and TAU constituents are both present in a particular example and there are myriad proxy constituents. Surveillance for a particular procedure is turned on and off by for every proxy constituent using the system implemented by the Mastermind constituent.

By default, the instantaneous proxy generator would then supervise all methodologies, but deactivating tracking is simple. As a result of TAU's features and functions, the Mastermind will be capable of measuring efficiency. Fig 4 and Fig 5 show illustrations of the interactions between the three types of components in the measurement tool. When looking at Figures 1 and 2, you can see that constituent C is already being integrated in both. According to Figure 4, one proxy function can be used to initialise multiple connections. As shown in Figure 5, component C can be configured to use completely separate proxies at each of the ports supplied by it. In combination with a fully automated proxy generation system, this is the layout. An application developer's workload is increased when using proxy generators, as shown by this example. Proxy management, on the other hand, is an easy operation, and only one proxy per constituent approach streamlines the unpredictability of the instantaneous proxy generator.

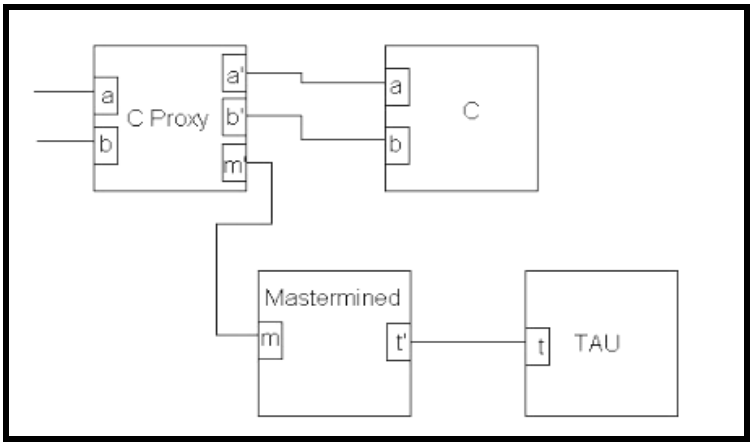

Fig 4. Measurement model using a single proxy in every constituent

It was decided to use the Efficiency Data Management Conceptual model to make the backup and evaluation of the experimental measurements easier. When used in this research, the plan was mostly used to store evaluation information in the database.

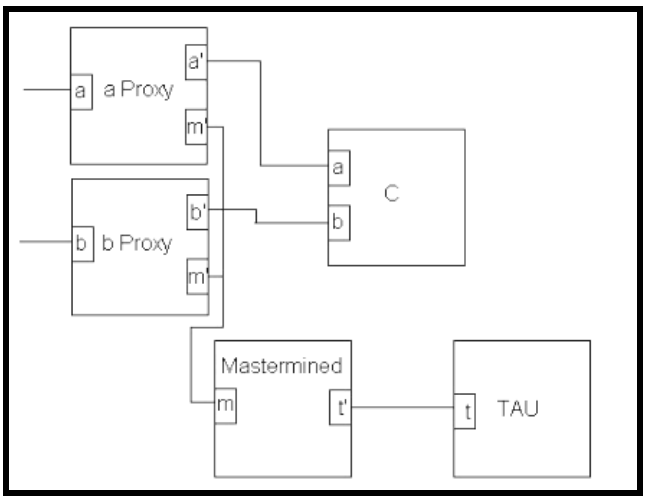

Fig 5. The measurement framework using one proxy in each port

Choice of Optimal Constituent Implement

An evidence-based performance model for individual elements can be engendered using a variety of techniques, including such multiple regressions, when constituent software is instrumented and performance information is recorded. To create a conceptual framework for the constituent application, these analytics system can be combined together. We are left wondering including one that constituent configurations are going to be most efficient in solving our problem. Selecting only the constituent with the highest efficiency components may not assure a globally optimal solution because the designs for individual elements do not consider interplay among constituents. Transforming data input for mismatching database systems is one example of this type of interplay. Assessing all feasible constituent combinations in the international model developed will yield an optimum solution. A workable solution that is estimated preferable is described in this section. Ten to twenty constituents are typical in a science-based application. There are 310 to 320 cumulative setups if each constituent has three insights. We cannot enumerate and evaluate all potential insights by brute force due to the large optimal solution.

As a result of this, most scientific applications use solver constituents to perform the majority of their work. Finding and optimizing these prominent essential parts allows for estimated optimal results. To make the solution space more workable, it is possible to remove "inconsequential" elements from the initialization step. An application should be made near optimum level by optimization of the main components and then integrating any integration of the alloying elements. As a result of this, even a poor implementation alternative for a potential agent would have very little implication on the software's performance.

IV. CONCLUSION AND FUTURE DIRECTIONS

An application infrastructure is proposed in this article in order to measure performance in computational surroundings in a non-intrusive manner. It is really able to quantify performance with proxy servers because they're simple and efficient. The scope of a framework's use, on the other hand, must be factored into the equation when modeling its performance. Recording input parameters that influence a framework's performance is essential. However, the question of which

parameter settings to retrieve remains unanswered. Additionally, a technique has been developed for determining a near optimal choice of constituent adoption and implementation. As a first step, inconsequential branches are removed from the constituent call-graph in order to identify the core elements. Familiarize yourself with all of the adoption and implementation of a single element in the diminished call-graph by grouping them into families.

Choosing a single instance out of each family and assessing its framework and inter-component communication results in a multilateral performance model that can be sequenced and reviewed. It is feasible to realize an entire, near optimal solution by implementing any of the small, inconsequential constituents that were removed in the first stage. There was also a confirmation technique postulated in order to check the framework's validity. "Dummy" constituents that emulate a known conceptual framework can be used to test how accurate a model is. As an example, components of a test procedure were shown to have two different implementations. Each component's optimised integration was determined by the input. It was relatively easy to verify the outcomes of the quantification and optimization stages by hand because dummy constituents executed known models.

With the measurement conceptual model and optimization catalog defined in this article, there seem to be a lot of possibilities for further research. Listed below are a few of these points of research.

- Performance Modeling: Due to the high-level nature of the performance models proposed in this research, they are limited to applications that run on a comparable cluster of servers. For example, modifying storage capacity would have an influence on regression coefficient. For optimal results, the processor frequency and cache prototype should be used to determine the coefficients to be used in the simulations. In the execution selection stage, another component of modeling and simulation is how the designs are defined. As of now, the models must be manually coded into the catalog and assessed by it. Performance frameworks can be imported from a document as an executable representation. This would be far more user-friendly and efficient for everyone. As an alternative, the frameworks could be expressed in Metadata or Xml files, with the optimal control library parsing the document at running time and constructing the expressions. Last but not least, the performance models do not take into account any dimensions of performance, as has been stated numerous times.

Determining which solver was fastest, even though it was less accurate, was the result of using the metaheuristic catalog. This information could be incorporated into the assessment of the optimization models in future research.

- Dynamic Execution Selection: Among Ccaffeine's exciting facts is the capacity to adaptively integrate modules at program execution. If one constituent execution will not really work well, it can be swapped out for another. Some preparatory work on an Optimizer constituent has already been carried out. While this constituent can swap components, it can also insert a proxy and update the Mastermind module's interconnection with it and any other components.

As part of the future work, the Optimization algorithm will be able to evaluate the actual performance of the existing constituents in comparison to the performance expectations. It is indeed possible to swap out any constituents that are not even functioning well and also have an alterative integration.

References

[1]. D. Kuck, "Productivity in High Performance Computing", *The International Journal of High Performance Computing Applications*, vol. 18, no. 4, pp. 489-504, 2004. Doi: 10.1177/1094342004048541.

[2]. N. Parlavantzas and G. Coulson, "Designing and constructing modifiable middleware using component frameworks", *IET Software*, vol. 1, no. 4, pp. 113-126, 2007. Doi: 10.1049/iet-sen:20060050.

[3]. M. Aldinucci, V. Cardellini, G. Mencagli and M. Torquati, "Data stream processing in HPC systems: New frameworks and architectures for high-frequency streaming", *Parallel Computing*, vol. 98, p. 102694, 2020. Doi: 10.1016/j.parco.2020.102694.

[4]. S. Kaliraj and A. Bharathi, "Path testing based reliability analysis framework of component based software system", *Measurement*, vol. 144, pp. 20-32, 2019. Doi: 10.1016/j.measurement.2018.11.086.

[5]. 2021. [Online]. Doi: https://www.sandia.gov/~jairay/Presentations/rtsam_revised.pdf. [Accessed: 20- Sep- 2021].

[6]. T. LIU, B. FAN, C. WU and Z. ZHANG, "Dataflow-Style Java Parallel Programming Model and Runtime Optimization", *Journal of Software*, vol. 19, no. 9, pp. 2181-2190, 2008. Doi: 10.3724/sp.j.1001.2008.02181.

[7]. B. Palmer, Y. Fang, V. Gurumoorthi, G. Hammond, J. Fort and T. Scheibe, "Developing a component-based framework for subsurface simulation using the Common Component Architecture", *Journal of Physics: Conference Series*, vol. 180, p. 012064, 2009. Doi: 10.1088/1742-6596/180/1/012064.

[8]. "Through a glass, warmly: Argonne nanomaterials can help make windows more efficient | Argonne National Laboratory", *Anl.gov*, 2021. [Online]. Doi: https://www.anl.gov/article/through-a-glass-warmly-argonne-nanomaterials-can-help-make-windows-more-efficient. [Accessed: 20- Sep- 2021].

[9]. G. Kumfert et al., "How the common component architecture advances computational science", Journal of Physics: Conference Series, vol. 46, pp. 479-493, 2006. Doi: 10.1088/1742-6596/46/1/066.

[10]. C. Parsons, "Inappropriate Access to the Adolescent Patient Portal and Low Rates of Proxy Account Creation", JAMA Network Open, vol. 4, no. 9, p. e2125251, 2021. Doi: 10.1001/jamanetworkopen.2021.25251.