Summarizing task-based applications behavior over many nodes through progression clustering

Lucas Leandro Nesi, Vinicius Garcia Pinto, Lucas Mello Schnorr, Arnaud Legrand

Summarizing task-based applications behavior over many nodes through progression clustering

Lucas Leandro Nesi
Institute of Informatics
PPGC/UFRGS
Porto Alegre, Brazil
Univ. Grenoble Alpes
Grenoble, France
lucas.nesi@inf.ufrgs.br

Vinícius Garcia Pinto
Computer Science Center
Federal Univ. of Rio Grande
Rio Grande, Brazil
vinicius.pinto@furg.br

Lucas Mello Schnorr
Institute of Informatics
PPGC/UFRGS
Porto Alegre, Brazil
schnorr@inf.ufrgs.br

Arnaud Legrand
Univ. Grenoble Alpes, CNRS,
Inria, Grenoble INP, LIG
F-38000 Grenoble, France
arnaud.legrand@imag.fr

***Abstract*—Visualization strategies are a valuable tool in the performance evaluation of HPC applications. Although the traditional Gantt charts are a widespread and enlightening strategy, it presents scalability problems and may misguide the analysis by focusing on resource utilization alone. This paper proposes an overview strategy to indicate nodes of interest for further investigation with classical visualizations like Gantt charts. For this, it uses a progression metric that captures work done per node inferred from the task-based structure, a time-step clustering of those metrics to decrease redundant information, and a more scalable visualization technique. We demonstrate with six scenarios and two applications that such a strategy can indicate problematic nodes more straightforwardly while using the same visualization space. Also, we provide examples where it correctly captures application work progression, showing application problems earlier and as an easy way to compare nodes. At the same time that traditional methods are misleading.**

***Index Terms*—HPC, Visualization, Performance Analysis, Task-Based, Heterogeneity**

I. Introduction

The performance analysis of High-Performance Computing applications is a vital step for achieving correct performance. However, the complexity of the applications and systems, including the many levels of heterogeneity [1], presents considerable challenges. There is also a popularity increase of programming paradigms like the Task-Based one, as it improves the portability of applications across different systems, enables easier asynchronous executions, and improves composability of the applications [2]. Such characteristics demand tailored behavior analysis tools and strategies to aid developers and analysts in their performance evaluation. In this way, the performance analysis of HPC applications through visualization is considered an advantageous methodology [3], as it enables a facilitated comprehension of large amounts of trace data [4].

However, even when using visualization, the comprehension of applications' performance may be shadowed by the number of features the system presents. This is particularly the case when using Gantt charts plotting states per node-level resources (cores, GPUs) with large amounts of nodes. Although there are techniques for aggregating and reducing the size of such visualizations [5]–[8], the traditional idea of having system elements as one axis would never scale as their number grows. Moreover, these Gantt chart ideas focus on resource utilization, which does not necessarily reflect how well the application is distributed and progresses. Aggregating many resources with the same utilization would also mask application problems and their progression.

A critical aspect of such analysis workflow is the progression of the application through its final result. Measuring at a given point how far the application is from its goal, from its ideal performance, and how each node behaves compared to each other can lead to performance improvement insights. We argue that an entry-level visualization for the performance analysis workflow that focuses on the progression would complement the Gantt chart by providing an overview. Such entry-level visualization should guide the analyst to points and nodes of interest by serving as a preliminary step before delving into the details of application behavior using the Gantt chart. This visualization should enable the perception of node outliers and handle heterogeneity in its many forms, such as resources, application phases, and tasks.

This paper studies and proposes a performance analysis methodology through an entry-level visualization for checking the progression of task-based applications on individual nodes to indicate moments and node groups of interest. This methodology comprises three elements: a progression metric per node that can be inferred from the structure of task-based applications, a clustering method to classify nodes and reduce the elements to show, and a final entry-level visualization of such components. The specific contributions of the paper are the following. (1) We study progress metrics and how they detect and represent performance behavior when exposed to different situations, system-level node heterogeneity, and multiple tasks. (2) We propose a strategy for clustering and visualizing such progression metrics to aid the identification of problematic groups of nodes and how much these groups are

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, with grant nº 88887.624965/2021-00 to the first author, the National Council for Scientific and Technological Development (CNPq), grant no 141971/2020-7 to the first author, INRIA, and the projects: FAPERGS (Data Science – 19/711-6), CAPES/Cofecub (04/2017).

performing behind compared to the performance lower bound references. (3) We perform a comprehensive analysis of such visualization in six situations with two different applications, using the Chameleon dense linear algebra library [9] over the StarPU runtime [10] and the ExaGeoStat application [11].

The paper is structured as follows. Section II presents the background and related work on performance visualization of HPC and task-based applications. Section III proposes the methodology for visualizing the progression of groups of nodes. Section IV demonstrates how this strategy discovers problems in different cases. Section V concludes the paper with a discussion and future perspectives. A publicly available companion[1] includes the paper's data and visualization code.

II. Behavior Visualization of HPC Applications

HPC Applications traditionally run on a large number of dedicated computing nodes. Identifying performance disturbances, bottlenecks, or computation progression in such scenarios is challenging, requiring more than general summarized metrics such as makespan, speed-up, bandwidth, and memory usage. The instrumentation of the source code to register relevant timestamped events enables later reconstruction of the execution behavior.

A common approach is to translate the logged events into a visual representation. Performance analysis visual representations come in different flavors as histograms, call-graphs, scatter plots, line charts, or treemaps [4], [12]. Regardless of this assortment, the most prevailing views are Gantt-like charts [13] where one axis is the time, and the other represents resources or entities with application states being mapped on such space.

Gantt-like charts, however, have some limitations. Regarding the platform, they do not scale with a large number of nodes, each one possibly exposing many workers (e.g., CPU cores, GPU cards). Considering the software, such charts do not allow us to easily distinguish the numerous hierarchical states disclosed by current applications (e.g., phases, iterations, low-level library kernels). Some tools [5]–[8] provide scaling to Gantt charts through aggregation, looking for similarity and uniformity to group similar states or behavior. Other scalability approaches rely on different representations such as treemaps [4], foldable graphs [14], or concurrent iterations [3].

A. Task-Based applications and their visualizations

In the task-based paradigm, the algorithm is structured in small portions of work (i.e., tasks). Each task has its parameters flagged as input, output, or input/output. In addition to the submission order, this allows the construction of a Directed Acyclic Graph (DAG) of tasks depicting their dependencies. From this graph, it is possible to identify and execute independent tasks in parallel. Task-based applications run on top of dynamic runtime systems [10], which are in charge of synchronizations, load distribution, and data transfers. In this work, we use the StarPU runtime system that can split the application workload (i.e., the DAG of tasks) among distributed nodes, each one running a local scheduler. This model enables fine-grained synchronizations, which favors workers' occupation and performance but increases the complexity of the performance analysis.

Some recent works [14]–[16] present task-based-oriented performance analysis features. They rely on task-based applications' DAG structure to provide scalable views through node folding/unfolding. StarVZ [3], [17], [18] offers different approaches that allow following the application' progress aside from the Gantt chart. It offers two progress views, one exposing the start/end time of iterations on iterative applications and the other exploiting the elimination tree structure used by sparse linear algebra solvers.

This work proposes a new visualization strategy that captures actual node progress based on its total scheduled workload. Our strategy relies on clustering to group nodes by their current performance enabling much better scalability than existing visualizations. This groping method quickly drives the analyst to problematic nodes, even when the traditional Gantt visualizations still do not disclose the performance issues. Aside from issues harming a few nodes, our new strategy also differs from existing ones because it can identify global performance disturbances affecting all nodes by comparing their current behavior with theoretical lower bounds.

III. Method Proposal: Node Progression visualization through clustering

As shown in the last Section, Gantt charts usually play a central role when analyzing HPC applications. However, when many resources have to be considered, the raw Gantt visualization is inadequate, and other summarizing techniques should be used. Instead of observing the Gantt chart or its variations as the first instrument, this Section presents a methodology for an entry-level tool to provide an overview. This overview can indicate and classify nodes of interest, where the traditional methods (including detailed Gantt charts) could use such groups to focus on non-redundant information.

A. Progression Metrics

The notion of how much work the application has accomplished at a given time, or how much it requires to finish, can be captured into a progression metric. When improving or analyzing an application's performance, its progression is an aspect of main interest, as if it progresses correctly and fast, the application's execution time will be optimized. In a distributed scenario with many resources, it is desired to understand how each node performs relatively.

We define a progression metric to capture the individual and normalized $[0, 1]$ progression of a node's work while maintaining the relative comparison possible. This comparison means that when nodes achieve the same relative progression (i.e., 50% of work), their metric should be the same.

The important aspect is the quantification of work. It can be directly associated with the application's main algorithm; for example, the iteration number concluded on a particular

[1]https://gitlab.com/lnesi/companion-pdp-2023

simulation step. However, it could be agnostic to the application and use information from the programming paradigm, in our case, the task-based one. In this agnostic case, universal information for all applications like tasks, the DAG, and task performance models can be used to measure work.

One very simple progression metric to be defined in the context of the task-based programming paradigm is the number of tasks completed $C_{s,n}$ at a given moment s in a node n as the primary measure of work and normalizing it by dividing with the node's (n) total amount of tasks N_n. The progression of a node n at a given time s, $P_{n,s}$ will be given by:

$$P_{n,s} = \frac{C_{n,s}}{N_n} \tag{1}$$

However, this simple metric works when the application has a clear dominant phase and task. Complex applications with multiple phases [19] and a variety of significant tasks would require quantifying different tasks costs considering the possible intra-node heterogeneity. This quantification could be achieved by using tasks duration $T_{n,t,r}$ of each heterogeneous resource r (CPU, GPU) in the node n to relativize them between task types t. The proposed metric is present on Equations 2 and 3. $W_{t,n}$ is essentially how much time a task t would hypothetically take if all node n resources with an available implementation could run it simultaneously, creating a weight per task type. Then, the progression ($P_{n,s}$) can be measured by summing for each task, the number of tasks completed $C_{n,s,t}$ of type t on node n at time s multiplied by the respective weight ($W_{t,n}$), and normalizing in the same way by replacing C by N. Such a metric would detect fewer longer tasks' progression and contribution correctly.

$$W_{t,n} = \frac{1}{\sum_r \frac{1}{T_{n,t,r}}} \tag{2}$$

$$P_{n,s} = \frac{\sum_t (C_{n,s,t} W_{t,n})}{\sum_t (N_{n,t} W_{t,n})} \tag{3}$$

To illustrate how this metric behaves, consider a case of the dense linear algebra Chameleon library with the LU factorization, where when using 30 nodes with two GPUs each, two nodes were faulty and were initialized with only one GPU. Figure 1 presents the aggregated Gantt chart per node resource of such execution made with StarVZ. Each node has two rows; the first is the aggregation utilization of the CPUs (four cores per node), while the second is for GPUs (two per node). In this case, it is possible to notice that two nodes (0 and 1) were working more than the others, and already with 30 nodes, such visualizations present scalability problems even with intra-node resource aggregation. The initial yellow tasks represent a generation data phase, and after synchronization, the green area represents the main computational task, the `gemm` operation.

Figure 2 presents the progression metric that accounts for heterogeneity for each node of Figure 1 execution. At the start of the execution, annotation A.3, there is a split in behavior. The below line (actually two overlapping lines) represents the

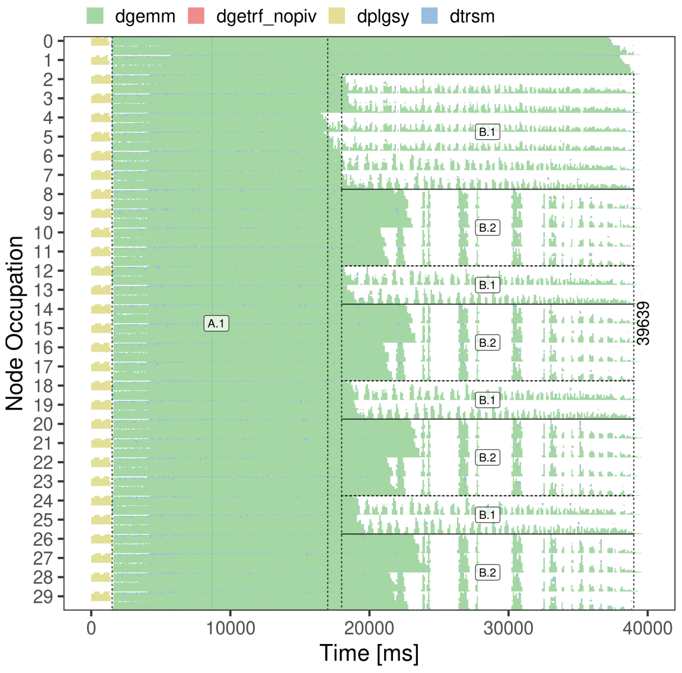

Fig. 1. Gantt chart with nodes' resources aggregation of the Chameleon LU Factorization execution with 30 nodes where two are misbehaving.

progress of the two slower nodes. The upper lines show the progression of the other nodes. In the middle of the execution, there is also a division of nodes in two distinct behavior, annotation B.3, observed from Gantt chart areas B.1 and B.2.

The progression metric, which is not a utilization metric (as in the Gantt chart), indicates that a group of nodes is progressing slower since the beginning, annotation A.3, a situation that is not clear in the Gantt area A.1. The Gantt gives a false impression in area A.1 that things were progression well, where actually there was a problem in the load partition since the start (considering the relative real speed of the machines). In the middle of the execution, the split in behavior is directly associated with the data partition, as will be discussed in Section IV. Where B.1 and B.2 are associated respectively with nodes that communicate directly or indirectly with the two problematic nodes. Also, this visualization already has a lot of lines (one per node).

B. Summarizing by Clustering

Analyzing such progression metrics may still be difficult in cases with many nodes having similar behavior, i.e., similar metric values. The identification of nodes or groups of nodes of interest can still be overwhelming. A possible solution is summarizing such metrics by clustering and understanding the behavior of node groups instead of individual ones. We utilize the notion of discrete time-steps, computing the progress metrics and clustering the nodes only for those moments. This clustering per step is particularly useful because some performance degradation may arise at lonely moments of the execution, and while at that particular moment, the node is an

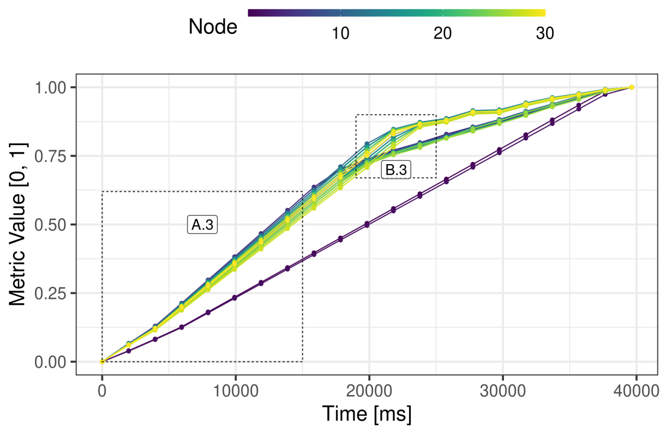

Fig. 2. Progression heterogeneous metric applied to the Chameleon simulation of the LU Factorization on 30 nodes.

outlier, during the rest of the execution, it behaves the same as other nodes.

At a given time-step, the progression may be summarized by clustering nodes that follow the intuitive, though weak and relative, characteristics: (1) Have the same progress behavior but have differences in the progression metric because of system variability, where one would expect a normal distribution; (2) Have genuine, though light, differences in behavior (small variances in workload, for example), but such small differences relatively do not reflect points of interest for the analyst. To comply with such characteristics, we select modal clustering, where the populations and clusters are defined as high regions of density divided by low regions of density [20]–[22]. Such a definition would encompass both characteristics.

We also tested K-Means and Gaussian Mixture Models (GMM) [23], but such approaches require determining the number of k clusters or distribution components. One of the most popular approaches to determine this is using BIC (Bayesian Information Criterion); however, it requires the number of observations to be large [24], [25]. In our case, we also may want to cluster groups with a few outlier nodes, even if this group comprises only one member. Also, for the case of the GMM, which models clusters as a realization of a normal distribution, we prefer generality that our groups are not necessarily normal distributions.

The modal clustering is performed with the principle of the mean shift algorithm [20], where data points are moved to the near kernel density modes and clusters are divided by local minimums [21], [22]. For this one-dimensional case with a relatively small number of data points, this can be achieved by computing the density estimation (using the Gaussian kernel in our case) with a bandwidth parameter h and finding the local minimums [22]. All data points between the same minimums are classified in the same group as belonging to that mode. Figure 3 presents the density estimations for the first ten time-steps with a bandwidth of 0.01, where the horizontal axis is the progression metric value, the vertical one is the density, and the blue vertical lines are the local minimums. At step one, all nodes share the same progression metric, zero, so it has a unique mode centered at 0. At step two, two nodes start to depart from the other 28; this causes a skewed distribution, with the mode much closer to the metric of the 28 nodes. But still, the distance between points using this bandwidth was not sufficiently far for creating another mode. At step three, there was enough distance, making two local modes, with a local minimum in the middle separating the groups. The left side represents the two slower nodes, while the right side is the other 28 nodes. The same principle continues through the remaining steps.

The advantages of this clustering method are that they work with a small number of data points, has a bandwidth parameter that controls the sensibility and the smoothing factor of the density estimation, enabling the adjustment of the clusters spread, and that clusters may be a complex mixture distribution that admits more complex density shapes than a normal distribution. The disadvantages of this clustering are that it does not capture possible sub-populations of interest that share the same mean but have different variances and the control parameter that must be adjusted. Also, this step-based clustering discards temporal knowledge that could be exploited.

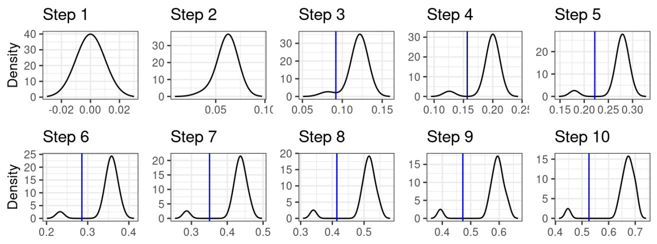

Fig. 3. Kernel density estimations of the first ten time-steps with the Gaussian kernel and bandwidth 0.01 for all steps considering metrics of Figure 2.

C. Progression Visualization

After having computed the clusters for each time-step, we propose a visualization to show the modal clustering progression. It is then shown in Figure 4 for the same data as Figure 2. At each time-step, one point is displayed per cluster. A line will connect two points of subsequent steps if they share nodes, and the number of shared nodes determines its thickness. In this way, from time 0 to time ~2000, all nodes are associated with the same cluster and share all nodes. However, at time ~2500, two clusters are detected, and the difference in thickness (thin on the bottom and thick on the top) informs that most of the nodes followed the up path behavior. This is the case as the upper cluster is made of two nodes while the below one is made of 28 nodes. At each disjoint path, the paths with fewer nodes have a label with the nodes that follow it. Also, the visualization keeps the original progression metrics per node (as of Figure 2) as background semi-transparent gray lines.

However, even if all nodes follow the same behavior, the application does not necessarily achieve the ideal performance.

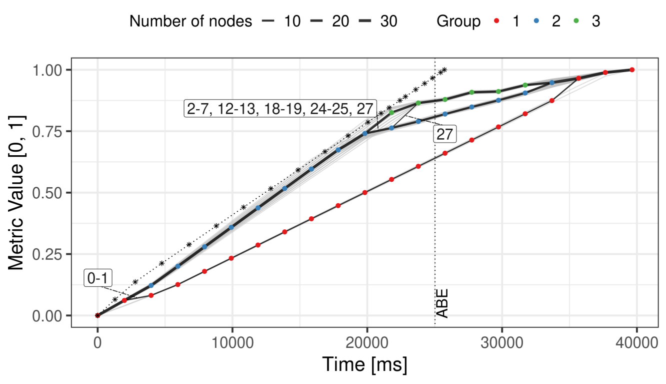

Fig. 4. Progression visualization strategy applied to the Chameleon simulation of the LU Factorization on 30 nodes.

All nodes could share the same problems and be equally late. We plot two additional metrics to aid the observation of such a problem (see Figure 4). First, the visualization presents the global Area Bound Estimation (ABE) [3] as a vertical dashed line. This is a lower bound when assuming that the duration of the tasks is not impacted. Second, we compute the ABE per time-step and perform a cumulative sum for each step of their ABE and their previous ones. A series of black points interconnected by a dotted line, usually in the upper part, demonstrates the cumulative per-step ABE. The first point is the ABE per step of the first step, while the second one is the ABE per step of the second step plus the first one, and so on. This metric captures some critical paths and steps' resource restrictions. For example, until step one, there are only tasks that do not utilize GPUs; the ABE of this step will only consider CPUs letting the GPUs idle. This is different from the global ABE, which will try to "pack" all tasks as if they do not have dependencies and compute the bound as if the GPUs were used in the early moments of the execution by future tasks. This difference explains why the per-step ABE may be longer than the global ABE. With those two metrics, one can have a better reference of the nodes' progress.

IV. Evaluation on Real Applications

This Section presents the adoption of our proposed methodology on real applications and how it could identify problems.

A. System and Software

The experiments were conducted with real executions and simulations (with StarPU-Simgrid [26], [27]) using Chameleon commit `54e4ec73`, StarPU commit `0fb603d8`, and Simgrid commit `61ee012f`. The version of ExaGeoStat, the application used in Section IV-C, is `9518886`. Simulation was used for the LU factorization experiments (Section IV-B) with a workload of size 96000×96000, considering 30 machines with eight cores (Intel Xeon E5-2620 v4) and 2 NVIDIA GTX 1080ti GPUs each. For the ExaGeoStat real experiments (Section IV-C), two partitions of machines were used with a 96100×96100 workload. First, six nodes with 32 cores each (2x AMD EPYC 7301); second, two nodes with 24 cores (2x Intel Xeon 6126), and two Nvidia P100 each. The modal clustering uses the `density` function from `R` 4.2.1 [28].

B. Chameleon predefined abnormal behaviors

Intending to test the methodology's identification power in common problems, we define a set of situations that may happen during the execution of real applications. Such situations are: (a) Communications contention problems of one or more nodes; (b) The utilization of a wrong distribution, giving more load to some nodes; (c) Global bad behavior that appears to be correct. Those situations were created using StarPU-Simgrid simulations and the Chameleon application. The next Sections detail these situations.

1) One node with slower connection: In this case, the first node has only a 1Gb/s network while the others have 25Gb/s. Figure 5 shows the progress clustering metric with 20 time-steps on the upper panel and the respective Gantt chart with nodes' resources aggregation on the lower one.

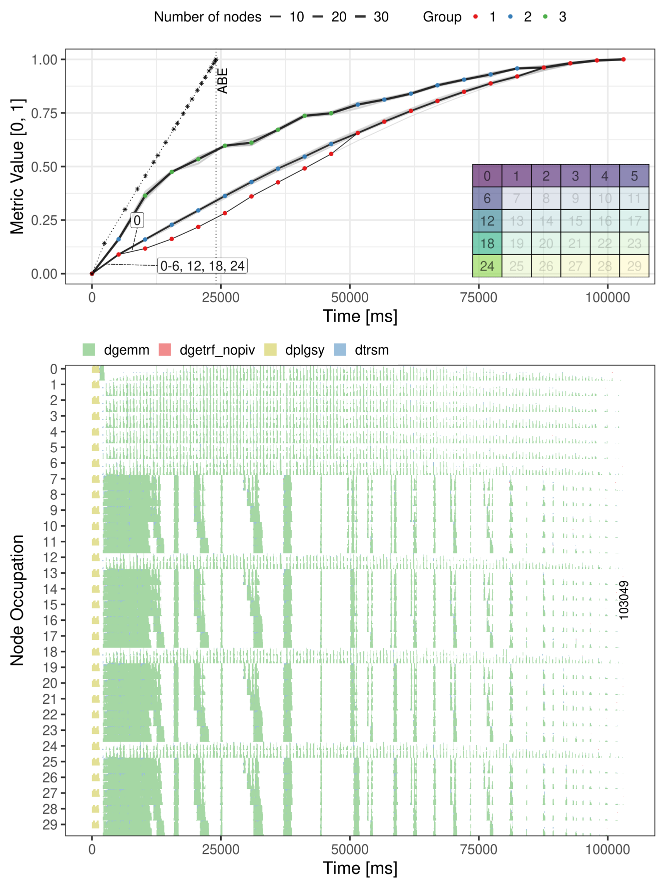

Fig. 5. Progression visualization strategy and aggregated Gantt chart of Chameleon's LU Factorization simulation over 30 nodes where the first node has a slower network.

Three groups of nodes dominate such execution. The slowest group, comprised of only one node (node zero), is exactly the node with the network reduction. The second slowest group is composed of nodes 1-6, 12, 18, and 24 and is only a little better than group one (the slowest). At the same time, the final and last group is composed of all remaining nodes. The distance from the lower bounds is significant, and while groups one and two started with the slowest progression since the beginning, group three followed the lower bound's progression until time ~15s. The explanation of this group split relies on the application matrix distribution.

The Chameleon library uses the traditional block-cyclic distribution [29] to divide its matrix across many nodes. Considering n nodes, this distribution works by setting two parameters p and q as $p \times q = n$. These parameters will be used to create a simple partition with p rows and q columns that will be used as a repetitive pattern through all the matrix block distributions. This partition matrix is depicted at the bottom right of the progression metric panel and presents the partition with $p = 5$ and $q = 6$ used in this case. In this distribution, considering the LU factorization kernel, nodes essentially communicate with other nodes that share the same row and columns. The nodes that share the row and columns with the problematic node zero are exactly the nodes of cluster two. This is not necessarily obvious, as one could expect one problematic node to cause a *global* slowdown in the whole system. However, the behavior of nodes that maintain direct communication with node zero is more affected. This observation corroborates that the clustering manages to group significant co-related nodes. From the performance analysis perspective, the nodes with a slower progression would have a priority in the analysis with more detailed tools such as Gantt charts. When observing the Gantt chart solely, it is not clear what is the group of most problematic nodes, though there is a distinction between the two groups in behavior. The progression metric informs such problematic nodes straightforwardly, with the benefit of being a more scalable visualization.

2) Bad distribution of load across nodes: In this case, we consider a heterogeneous distribution (1D-1D [30]), giving 50% more load to the first node and 25% more load to the second node. Figure 6 presents the progression metrics for this execution in the upper panel with the partition of this heterogeneous distribution on the bottom right. The bottom panel is the aggregated Gantt chart. Nodes zero and one have larger areas related to the increase in load (though they have the same computational power as the others). There is an expectation that this execution would have problems, but we are interested in checking how the nodes clusters relate to these problematic nodes.

Groups one and two from time 8s to 24s are solely the nodes zero and one, the problematic ones. There are five clusters at a maximum on time of ~24s. Group three is essentially nodes 2-5, 10, 15, 20, and 25, while group four is nodes 6, 11, 16, 21, 26, and group five is the others. This corresponds to the nodes that directly communicate with the problematic nodes, with a small difference between groups three and four, that the latter communicates more with node one (25% slowdown). While the groups can be recognizable in the Gantt chart, it is unclear to indicate which ones are the most affected. Another situation is that most nodes get affected by the slowdown only at the 20s-25s, progressing until then near the bound.

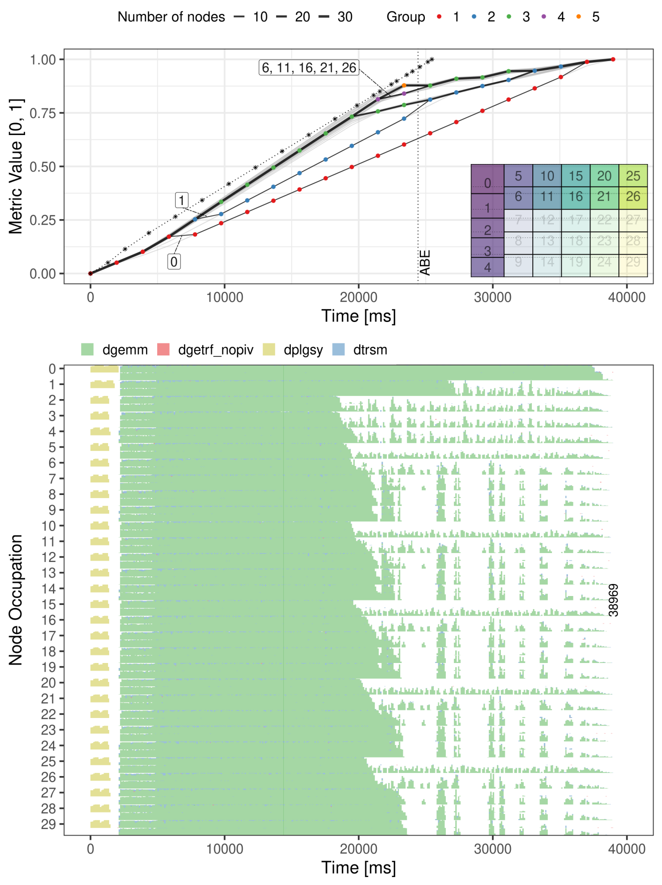

Fig. 6. Progression visualization strategy and aggregated Gantt chart of Chameleon's LU Factorization simulation over 30 nodes where the first and second nodes received 50% and 25% more load, respectively.

3) All nodes with bad network: The network can also globally impact the performance of the application. This case considers a scenario where the infrastructure has not an optimal network. In this case, all nodes have a 1Gb/s network. Figure 7 left presents the clustering metrics for this case. All nodes share the same cluster and present similar behavior. This could lead to the analysis that the execution was well performed. However, the metrics distance to the lower bounds of per-step and global ABE indicates a problem in execution.

The problem of Figure 7 left execution is even more discernible when it is compared to a correct execution without problems, as shown in Figure 7 right side. In the correct case, it is possible to check the proximity of the metric to the lower bound, indicating a well-behaved execution. The correct execution also demonstrates the difficulties of adhering to both ABE bounds at the end, when parallelism diminishes.

C. A multi-phase application over heterogeneous nodes

ExaGeoStat is a machine learning application that models Geostatistics with the Gaussian Process [11]. ExaGeoStat has five phases, the two most prominent phases being data generation and Cholesky factorization.

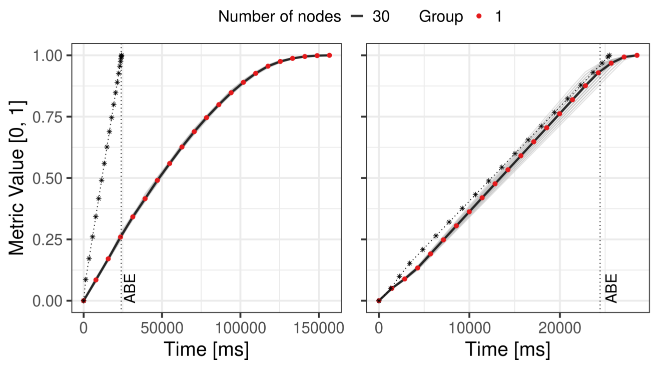

Fig. 7. Progression visualization strategy of Chameleon's LU Factorization simulation over 30 nodes where, on the left, all nodes have slow networks, and on the right, regular networks.

In this case, we use eight nodes, where six are CPU-only, and two have CPUs and two GPUs. Figure 8 presents the progress cluster visualization on the top panel and the aggregated Gantt chart on the bottom. The first phase is depicted in the Gantt by the yellow tasks and in the A area on the top panel.

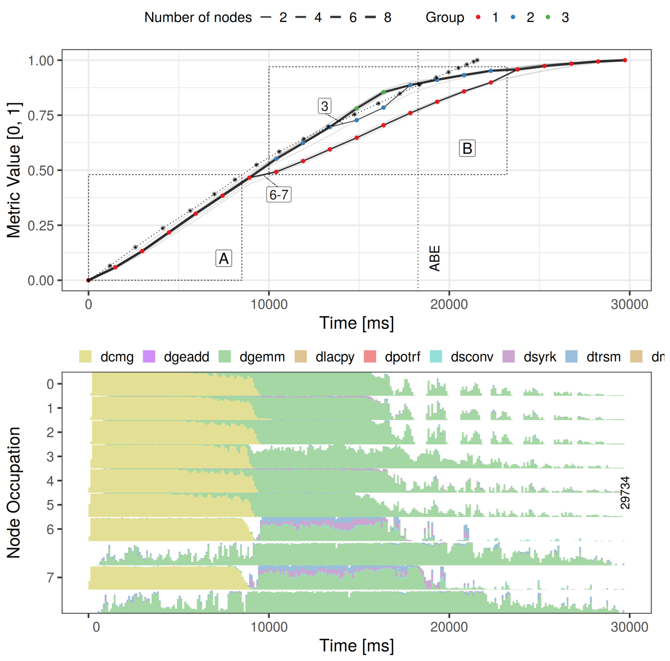

Fig. 8. Progression visualization strategy (with a bandwidth of 0.15) and aggregated Gantt chart of ExaGeoStat real execution iteration on eight heterogeneous nodes.

After the first phase, the nodes split into two clusters, despicted in the B area, and for a short period, three clusters. The fastest cluster (the top one) comprises six nodes that are exactly the nodes of the first partition, while the slowest cluster (in red) is the two nodes of the second partition (six and seven). With this indication from the progression visualization that some nodes were not performing correctly, further investigation is performed into them. In this case, the slow performance is caused by mainly two problems. First, we used distributions that consider the machines' relative power as inputs for this heterogeneous execution. The mean duration of all tasks on all nodes is considered for computing such power. However, when generating such distribution, there was an overestimation of the performance of the main task, dgemm, on GPUs by 5%. Second, this power considers a continuous utilization of the GPUs. However, in this case, these workers had short idle times between many tasks, accounting for 9% idle time. The StarPU scheduler used, *dmdas*, first greedy considers priorities, then the locality of data of ready tasks. A task becomes ready when its data is available on that node, i.e., dependencies are met and transferred to that node via MPI. Because of data of high-priority tasks arriving on this node, the scheduler will not be able to pre-fetch such data and will immediately schedule such high-priority tasks above all others, even if their data is still in RAM and have to be transferred to the GPU. Although there is a pipeline of tasks on GPU workers, the transfer duration is higher than the duration of the tasks, and the GPU will wait for the transfer to finish before starting the high-priority task.

This case also illustrates how the Gantt chart can be misleading directly. Figure 9 shows the Gantt chart for all resources for this execution (of Figure 8) for a middle point in the execution for node 0 (Gantt chart behavior equal to nodes 1, 2, 4, 5), node 3, the one that formed a new cluster between the fastest and slowest one, and node 6 (Same behavior as node 7). When looking at this Gantt, one may conclude that there is a huge problem on node three because of CPU workers' idle times. However, it is difficult to notice the small GPU idle times that account for a considerable loss in overall performance, as described before. The idle times on node 3 are mainly critical paths of late tasks of nodes six and seven that increased communication contention.

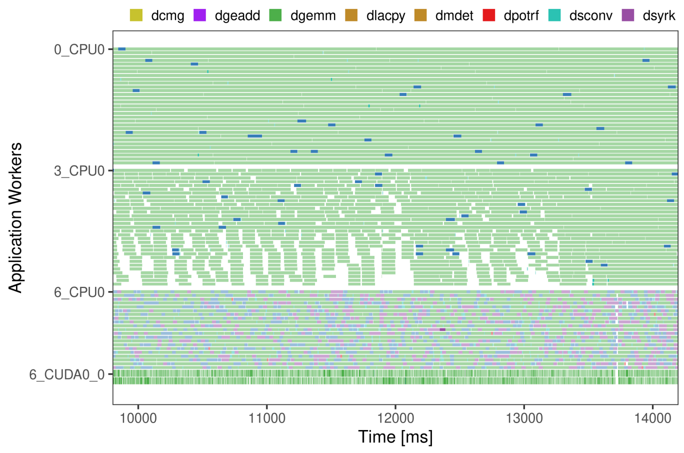

Fig. 9. Traditional Gantt chart of selected nodes of Figure 8 execution.

V. Conclusion and Discussion

Performance analysis of complex HPC applications is the core for improving and accelerating them even more. However, such performance analysis is difficult, as both applications and platforms present many levels of complexity, like heterogeneity, different phases, and sizes. Visualization can aid in this situation by quickly assisting in interpreting the application behavior. However, even the best visualizations have limited space for representing such amounts of data. Gantt charts are a classical visualization approach for such analysis. However, they are not scalable to the number of nodes and may point in the wrong direction. Although Gantt charts help visually identify resource idleness, they fail to determine or pinpoint the root cause of such inactivity that might be located elsewhere because task dependencies are much more complex in task-based systems.

This paper, intending to present a summary of application behavior and quickly capture the progression of nodes and indicate problematic ones, presents an entry-level strategy to provide an overview of the execution before using other methods, like Gantt charts. This strategy utilizes a progression metric to capture the behavior of the nodes, a clustering of such progression metric to identify nodes groups of interest and reduce the number of elements to show, and a visualization of such clustering over execution time. We evaluate such strategies over four crafted problematic scenarios with the dense linear algebra library Chameleon, which correctly detected the group of nodes with problems. In a real case with the ExaGeoStat application, it handled heterogeneity and indicated the most problematic nodes more straightforwardly than a traditional Gantt Chart. All of these results show the potential of such strategies. Future work includes the study of other progression metrics and clustering algorithms that may be tailored for some particular situations and the identification of clusters considering the temporal perspective.

Acknowledgements

Some experiments were carried out using Grid'5000, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (https://www.grid5000.fr). The authors acknowledge the National Laboratory for Scientific Computing (LNCC/MCTI, Brazil) for providing HPC resources of the SDumont supercomputer, which have contributed to the research results reported in this paper (http://sdumont.lncc.br).

References

[1] U.-U. H. HPE, E. Laure, S. Narasimhamurthy, and E. Suarez, "Heterogeneous high performance computing," *ETP4HPC White Paper*, 2022.

[2] O. Aumage, P. Carpenter, and S. Benkner, "Task-based performance portability in hpc," *ETP4HPC White Paper*, 2022.

[3] V. Garcia Pinto *et al.*, "A visual performance analysis framework for task-based parallel applications running on hybrid clusters," *Conc. and Comp.: Practice and Experience*, vol. 30, no. 18, p. e4472, 2018.

[4] L. M. Schnorr and A. Legrand, "Visualizing more performance data than what fits on your screen," in *Tools for High Performance Computing 2012*. Berlin, Heidelberg: Springer, 2013, pp. 149–162.

[5] V. Pillet, J. Labarta, T. Cortes, and S. Girona, "PARAVER: A Tool to Visualize and Analyze Parallel Code," in *Proceedings of WoTUG-18: Transputer and occam Developments*, P. Nixon, Ed., 1995, pp. 17–31.

[6] L. Schnorr, "Pajeng (paje next generation)." [Online]. Available: http://github.com/schnorr/pajeng/

[7] D. Dosimont, R. Lamarche-Perrin, L. M. Schnorr, G. Huard, and J.-M. Vincent, "A spatiotemporal data aggregation technique for performance analysis of large-scale execution traces," in *2014 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, Sep 2014.

[8] S. A. Sakin *et al.*, "Traveler: Navigating task parallel traces for performance analysis," 2022.

[9] E. Agullo *et al.*, "Faster, cheaper, better – a hybridization methodology to develop linear algebra software for GPUs," in *GPU Computing Gems*, W. mei W. Hwu, Ed. Morgan Kaufmann, Sep. 2010, vol. 2.

[10] C. Augonnet *et al.*, "StarPU: A unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice Experience, SI: EuroPar'09*, vol. 23, pp. 187–198, 2011.

[11] S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes, "Exageostat: A high performance unified software for geostatistics on manycore systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 12, pp. 2771–2784, 2018.

[12] K. E. Isaacs, A. Giménez, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, B. Hamann, and P.-T. Bremer, "State of the art of performance visualization," in *EuroVis - STARs*. The Eurographics Association, 2014.

[13] J. M. Wilson, "Gantt charts: A centenary appreciation," *European Journal of Operational Research*, vol. 149, no. 2, p. 430–437, Sep 2003.

[14] A. Huynh, D. Thain, M. Pericàs, and K. Taura, "Dagviz: A dag visualization tool for analyzing task-parallel program traces," in *Proceedings of the 2nd WS on Visual Performance Analysis*. ACM Press, 2015.

[15] R. Keller, S. Brinkmann, J. Gracia, and C. Niethammer, "Temanejo: Debugging of thread-based task-parallel programs in starss," *Tools for High Performance Computing 2011*, p. 131–137, 2012.

[16] N. Reissmann and A. Muddukrishna, "Diagnosing highly-parallel openmp programs with aggregated grain graphs," *Lecture Notes in Computer Science*, p. 106–119, 2018.

[17] L. L. Nesi, S. Thibault, L. Stanisic, and L. M. Schnorr, "Visual performance analysis of memory behavior in a task-based runtime on hybrid platforms," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2019, pp. 142–151.

[18] V. G. Pinto, L. Leandro Nesi, M. C. Miletto, and L. Mello Schnorr, "Providing in-depth performance analysis for heterogeneous task-based applications with starvz," *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Jun 2021.

[19] L. L. Nesi, A. Legrand, and L. M. Schnorr, "Exploiting system level heterogeneity to improve the performance of a geostatistics multi-phase task-based application," in *50th International Conference on Parallel Processing*, ser. ICPP. New York, NY, USA: ACM, 2021.

[20] K. Fukunaga and L. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," *IEEE Transactions on Information Theory*, vol. 21, no. 1, pp. 32–40, 1975.

[21] Y.-C. Chen, C. R. Genovese, and L. Wasserman, "A comprehensive approach to mode clustering," *Electronic Journal of Statistics*, vol. 10, no. 1, pp. 210 – 241, 2016.

[22] J. E. Chacón, "A Population Background for Nonparametric Density-Based Clustering," *Statistical Science*, vol. 30, no. 4, p. 518, 2015.

[23] ——, "Mixture model modal clustering," *Advances in Data Analysis and Classification*, vol. 13, no. 2, pp. 379–404, 2019.

[24] R. E. Kass and L. Wasserman, "A reference bayesian test for nested hypotheses and its relationship to the schwarz criterion," *Journal of the american statistical association*, vol. 90, no. 431, pp. 928–934, 1995.

[25] C. Giraud, *Introduction to high-dimensional statistics*. Chapman and Hall/CRC, 2021.

[26] L. Stanisic *et al.*, "Faithful performance prediction of a dynamic task-based runtime system for heterogeneous multi-core architectures," *Conc. and Comp.: Practice and Experience*, vol. 27, no. 16, p. 4075, 2015.

[27] L. L. Nesi, L. M. Schnorr, and A. Legrand, "Communication-aware load balancing of the LU factorization over heterogeneous clusters," in *26th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2020*. Hong Kong: IEEE, 2020, pp. 54–63.

[28] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2022. [Online]. Available: https://www.R-project.org/

[29] L. S. Blackford *et al.*, *ScaLAPACK User's Guide*. USA: Society for Industrial and Applied Mathematics, 1997.

[30] O. Beaumont, A. Legrand, F. Rastello, and Y. Robert, "Static LU decomposition on heterogeneous platforms," *Int. Journal of High Performance Computing Applications*, vol. 15, pp. 310–323, 2001.