Granula: Toward Fine-grained Performance Analysis of Large-scale Graph Processing Platforms

Wing Lung Ngai
VU Amsterdam, the Netherlands
w.l.ngai@vu.nl

Tim Hegeman
Delft University of Technology, the Netherlands
t.m.hegeman@student.tudelft.nl

Stijn Heldens
University of Twente, the Netherlands
s.j.heldens@utwente.nl

Alexandru Iosup
VU Amsterdam and the Delft University of Technology
a.iosup@vu.nl

ABSTRACT

Big Data processing has become an integral part of many applications that are vital to our industry, academic endeavors, and society at large. To cope with the data deluge, existing Big Data platforms require significant conceptual and engineering advances. In particular, Big Data platforms for large-scale graph processing require in-depth performance analysis to continue to support the broad applicability of linked data processing. However, in-depth performance analysis of such platforms remains challenging due to many factors, among which the inherent complexity of the platforms, the limited insight provided by coarse-grained "black-box" and inefficiency of fine-grained analysis, and the lack of reusability of results. In this work, we propose *Granula*, a performance analysis system for Big Data platforms that focuses on graph processing. *Granula* facilitates the complex, end-to-end processes of fine-grained performance modeling, monitoring, archiving, and visualization. It offers a comprehensive evaluation process that can be iteratively tuned to deliver more fine-grained performance information. We showcase with a prototype of *Granula* how it can provide meaningful insights into the operation of two large-scale graph processing platforms, Giraph and PowerGraph.

1 INTRODUCTION

Our society generates and processes increasingly more massive datasets, which are part of the worldwide digital universe already amounting to zettabyte [17]. Extracting useful knowledge from these large and complex datasets is one of key drivers of growth in the digital economy [10], but requires significant advances in the performance and efficiency of Big Data processing tools. To enable these advances, understanding the performance of a Big Data system is a crucial but challenging step. Coarse-grained performance analysis is useful for high-level system comparison and benchmarking [18], but cannot explain choke-points [2] or indicate which parts of the system can be improved. Fine-grained performance analysis is crucial in debugging, tuning system performance, and in finding points that require new system designs, but remains challenging and costly at scale even for non-data-intensive applications [4]. In this work, we propose our vision for and show preliminary experimental results obtained with *Granula*, a framework for fine-grained performance analysis of one specific but widely used class of large-scale Big Data processing systems.

Large-scale graph processing is a type of Big Data processing that specializes in the processing of massive graphs. A graph is typically defined as a data structure consisting of a collection of vertices and a set of edges connecting the vertices. As edges can occur between any two arbitrary vertices, the workload of graph application is highly irregular and imposes challenges in parallelizing and distributing the workload into multiple machines that do not appear in regular workloads. Thus, large-scale graph processing is one of the more challenging Big Data domains.

Due to the rapid development of and the large variations between Big Data platforms, it is challenging for users to compare and select the platform most suitable for their needs, or even for one application. Currently, tens of specialized Big Data platforms focus on diverse aspects of large-scale graph processing [7], e.g., Apache Giraph [6] on scalability, PowerGraph [19] on graphs with power-law distributions vertex-/edge-degrees, Oracle PGX.D [16] on the capabilities of powerful resources, and Intel GraphMat [24] on the similarities between graph processing and linear algebra. They also take widely different implementation approaches, as indicated by our summary in Table 1 and exemplified in Section 3.4.

Even when a Big Data platform has been selected, incorrect configuration or poor use can lead to performance slowdowns of several orders of magnitude [14, 23]. General Big Data platforms, such as the MapReduce-based *Apache Hadoop* [25], have not been able so far to process graphs without severe performance penalties [14, 20, 23]. Specialized platforms can also raise intricate configuration problems [14], and tuning challenges that vary across applications and datasets [18].

Performance evaluation is the process of deepening the understanding of the performance of these Big Data platforms by quantifying system performance, explaining performance differences, identifying overheads/bottlenecks, and recommending configuration improvements. We argue that the current state of performance evaluation of Big Data platforms in general, and of graph-processing platforms in particular [18, 20, 23], does not completely accomplish these goals. There are non-trivial issues in focusing on the complex architecture of each platform; in enabling the analyst to control the trade-offs between the fast, coarse-grained, "black-box" performance analysis, and the costly, fine-grained analysis; in analyzing data-processing end-to-end; and in sharing performance results for the entire community of analysts. Toward addressing these issues, the main contributions of this work are threefold:

(1) We identify four main issues in the performance evaluation of Big Data platforms (Section 2).
(2) We present the design of *Granula*, a performance evaluation system for Big Data (graph-processing) platforms

Table 1: Diversity in (large-scale) graph processing platforms. The systems in bold are the focus of experiments in this article.

Name	Vendor	Vers.	Lang.	Distr.	Provisioning	Programming Model	Data Format	File Sys.
Giraph [6]	Apache	1.2.0	Java	yes	Yarn	Pregel	VertexStore	HDFS
PowerGraph [13]	CMU	2.2	C++	yes	OpenMPI	GAS	Edge-based	local/shared
GraphMat [1]	Intel	-	C++	yes	Intel-MPI	SpMV	SpMV	local/shared
PGX.D [16]	Oracle	-	C++	yes	Native, Slurm	Push-pull	CSR	local/shared
OpenG [22]	Georgia Tech	-	C++/CUDA	no	Native	CPU/GPU	CSR	local
TOTEM [12]	UBC	-	C++/CUDA	no	Native	CPU+GPU	CSR	local
Hadoop [25]	Apache	-	Java	yes	Yarn	MapRed	Out-of-core	HDFS

(Section 3). *Granula* facilitates performance modeling, monitoring, archiving, and visualization.

(3) We conduct fine-grained performance evaluation for two large-scale graph processing platforms (Section 4). By focusing on significantly different platforms, Giraph and PowerGraph, we show evidence that *Granula* uses a generic evaluation process.

2 ISSUES IN PERFORMANCE EVALUATION FOR BIG DATA PLATFORMS

We identify four main issues in the current performance-evaluation approaches for Big Data platforms, which in our view need to be addressed systematically and comprehensively through approaches such as our *Granula*:

1. **Lack of an end-to-end evaluation process:** Overall, without a well-defined, standardized evaluation process, analysts need to redesign from scratch important or even all steps of the process: analyzing the Big Data platform, collecting metrics from the environment, storing data in specific format, and visualizing the final results. Different designs can lead to widely different processes, and even incomparable results.

2. **Limited reusability of performance studies:** A performance study can only be considered widely applicable when its results provide fundamental insights into system performance, and/or its evaluation process can be easily applied in different scenarios. However, in practice, performance studies are often out-dated quickly, as the Big Data ecosystem is rapidly evolving with new systems and updates [7]. Most performance studies are not easily reproducible and the results are not directly comparable. As a result, developers and users cannot fully benefit from these performance studies.

3. **Shortcomings in coarse-grained evaluation:** Many studies [14, 22, 23] consider a Big Data platform as a "black-box": by varying the input parameters (e.g., data size, type of algorithms, configuration) and by observing the corresponding changes in outputs (e.g., the execution time, resource utilization, cache hits), the analysts derive general, benchmarking-like conclusions on the performance of these platforms. However, overall outputs are poor indicators of where new concepts or good implementation can be most effective, because such outputs are consequences of many hidden internal operations. Neglecting the internal architecture of these platforms, the coarse-grained approach can only quantify, but not explain the performance differences between platforms.

4. **Inefficiency of fine-grained evaluation:** To gain a comprehensive understanding, the analysts could resort to fine-grained performance evaluation, for analyzing and quantifying the performance of the internal operations of Big Data jobs. However, conducting fine-grained performance evaluation requires much time and many (specialized) resources: in-depth and specific knowledge of each platform, effort to set up the experimental environment, occupation of computing and data infrastructure during experiments, and generation of large volumes of empirical data than must later be processed and further understood. Often, the effort required to conduct full-platform fine-grained evaluation cannot be justified.

3 *GRANULA*: VISION AND DESIGN

In this section, we present our vision for *Granula*, a fine-grained performance evaluation system for Big Data platforms.

3.1 Requirements

Comprehensive performance evaluation is challenging, due to the high complexity and diversity of Big Data platforms. To mitigate the significant yet inevitable investment in time and effort required to evaluate system performance, and allowing analysts to navigate the issues presented in Section 2, the design of *Granula* needs to satisfy the following requirements:

(R1) Comprehensive: Support an end-to-end evaluation process. (This addresses Issue 1.)

(R2) Standardized: Facilitate a reusable procedure across different platforms, so that the evaluation process is reproducible and the results are comparable. (Issue 2)

(R3) Incremental: Provide an iterative procedure that allows performance study to be done incrementally, balancing between the investment of effort and the comprehensiveness of results. (Issues 3-4)

(R4) Automatable: Reduce the amount of technical work by providing an implementation that automates the repetitive procedures in the evaluation process.

3.2 Performance Model (for Big Data platforms)

To study system performance, *Granula* applies a modeling language (see Figure 1) to abstract the complex operations and their relationships, which is designed to be generically applicable (requirement R2) to many Big Data platforms. The modeling language describes a Big Data job as a hierarchy of operations; a job at the top level of the performance model, consisting of multiple internal operations which are recursively decomposable into underlying operations. This enables expressing Big Data jobs as a complex flow of operations consisting of multiple stages, running on multiple machines.

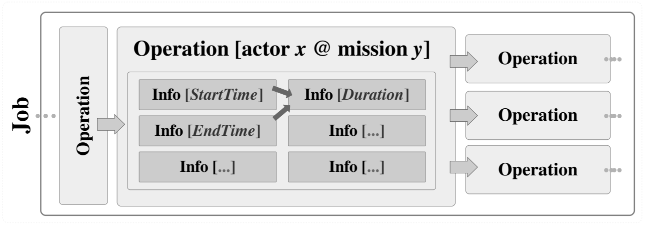

Figure 1: The *Granula* performance model.

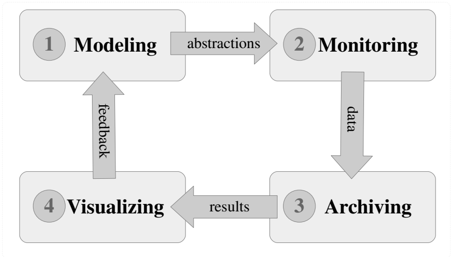

Figure 2: Overview of the *Granula* evaluation process.

Operations are the most important concept in the performance models developed with *Granula*. Each operation is annotated as an actor (e.g., resource) executing a mission (e.g., a computational algorithm, a communication protocol). *Granula* supports explicit description of operational patterns typically found in Big Data platforms, including task parallelism (multiple actors executing the same mission) and iterative processing (a mission being executed repeatedly by the same actor). Internally, the performance characteristics of each operation are described by its information set (*info*), which can be used to derive sophisticated performance metrics. Externally, the role of each operation in the platform is described by its links to the parent operation, and/or to the filial operations.

By using a top-down approach, analysts are able to study a Big Data platform and develop its performance model incrementally (R3), from coarse-grained to fine-grained, focusing only on system-components of interest. Analysts can build incrementally a performance model for each platform, to express their increasing understanding of system operations and to focus on the key performance issues. Although models depend on the vision of their creators, we propose that each platform could be modeled with at least three *levels of abstraction*: the domain level (1st), the system level (2nd), and the implementation level (3rd and *finer-grained*). The *domain level* summarizes the common elements in a particular domain, i.e., graph processing. The *system level* describes the operation workflow of each platform, allowing analysts to pin-point which system operations are suffering from performance issues. Finally, the *implementation level* reflects various implementation details, demonstrating the effectiveness of optimization techniques and showing the actual causes of performance issues.

3.3 Evaluation Process

To support a comprehensive end-to-end evaluation process (R1), the key elements of the evaluation process are formalized as four standard, consecutive sub-processes (R2): modeling, monitoring, archiving and visualization (Figure 2).

- **(P1) Modeling:** First, analysts study the design and the implementation of a platform, and express their understanding in the form of a performance model (Section 3.2), an abstract representation of the system's operations and performance characteristics. In the performance model, analysts define (1) the relationships between operations inside a job, (2) the set of raw data to be collected from the platform, and (3) the rules to transform raw *info* into performance metrics.
- **(P2) Monitoring:** Then, analysts set up an experimental environment to run platform jobs and monitor the system, gathering *info* as defined in the performance model. Two types of performance data are collected: (1) platform logs reveal the internal operations of the platform; (2) environment logs reveal the performance impact on the underlying cluster environment.
- **(P3) Archiving:** After experiments, the *info* of each job is collected, filtered, and stored in a performance archive with a standardized format. This performance archive encapsulates the performance results of each job, and allows users to query the contents systematically.
- **(P4) Visualization:** Finally, the archived performance results are presented in human-readable visuals, which allows efficient navigation and presentation of the results among analysts.

Knowledge about a Big Data platform's performance is built incrementally (R3). In each iteration of the evaluation process, analysts first study the platform and update the performance model with the insights gained during the previous iteration. Then, they execute platform jobs and acquire new performance archives. By reviewing and analyzing the resulting visuals, new insights in the platform's performance may be gained. In each iteration, analysts focus only on the system components that need finer-grained analysis, refining at most a subset of the model.

The entire evaluation process is implemented and modularized in *Granula*, which allows the process to be highly automated (R4): analysts only need to focus on the conceptual analysis (modeling), while *Granula* performs the repetitive technical tasks (monitoring, archiving, and visualization).

3.4 Large-scale Graph Processing

Showing how *Granula* can evaluate the performance of distributed graph processing platforms allows us to demonstrate the usability of *Granula* for a concrete application-domain. Many graph-processing platforms already exist, covering a broad set of characteristics [7]. Table 1 compares 7 platforms widely used in academia and practice, across 8 high-level characteristics. For example, **Giraph** [6] uses an iterative vertex-centric programming model similarly to Google's Pregel, is designed for generic graphs, and is implemented on top of Apache Hadoop's compute- and storage-management systems. **PowerGraph** [13] uses a programming model known as Gather-Apply-Scatter (GAS), is designed for real-world graphs which have a skewed power-law degree distribution, and does its own resource management.

To cope with platform diversity, we focus with *Granula* on the core process of any large-scale graph processing system. To process

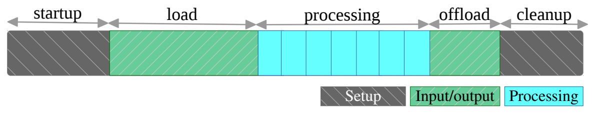

Figure 3: A high-level breakdown of a graph processing job.

graph data, users implement algorithms on a graph processing platform and run graph-processing jobs on a specific dataset. Each job can consist of tens or more operations, which are hidden to the analyst unless explicitly modeled. We categorize these operations into three types: setup, input/output, and processing operations (See Figure 3).

1. **Setup/Startup operations** reserve computational resources in distributed environments and prepare the system for operation. For example, Yarn-based platforms (e.g., Giraph and Hadoop) communicate with the Yarn manager to allocate resources, and MPI-based systems (e.g., PowerGraph, GraphMat) require an MPI cluster to use multiple nodes. Single-node platforms (e.g., OpenG, TOTEM) do not require any underlying resource manager other than the operating system. We denote setup time by T_s.

2. **Input/output operations** transfer graph data from storage to the memory space, and convert the data to specific formats before/after data processing. Some platforms load data from a distributed file system (such as HDFS [5] for Apache Giraph and Hadoop). Other distributed platforms load data from local storage in each node. Conversion of the input data format is also classified as an I/O operation. We denote I/O time by T_d.

3. **Processing operations** take in-memory data and process it according to an user-defined algorithm and its expression in a programming paradigm. For example, Giraph is based on the Pregel model [21], PGX.D allows both pushing and pulling of data, and GraphMat processes graphs as sparse matrices [26]. We denote processing time by T_p.

4 EXPERIMENTAL RESULTS

We conduct for this work experiments with *Granula*, focusing on *Granula*'s ability to support performance analysts in studying the performance of Big Data platforms, in particular graph processing platforms. We take with *Granula* a systematic approach for each platform under analysis (system under test): (1) develop a performance model to understand the system operations, (2) break down the job performance to quantify the performance of the fine-grained operations inside each job, (3) analyze the results by mapping resources usage pattern to the corresponding operations, and (4) visualize the internal system behaviours to identify hidden performance issues.

In this work, we compare two platforms: Giraph and PowerGraph. For each, we run the *BFS* algorithm on *dg1000* (a large Datagen [9] graph with 1.03 billion vertices and edges), using 8 compute nodes on the DAS5 Dutch supercomputer [3].

4.1 Building Performance Models

Using the concepts and following the levels introduced in Section 3.2), we have built *Granula* performance-models for various type of graph processing platforms, e.g., Giraph and Powergraph.

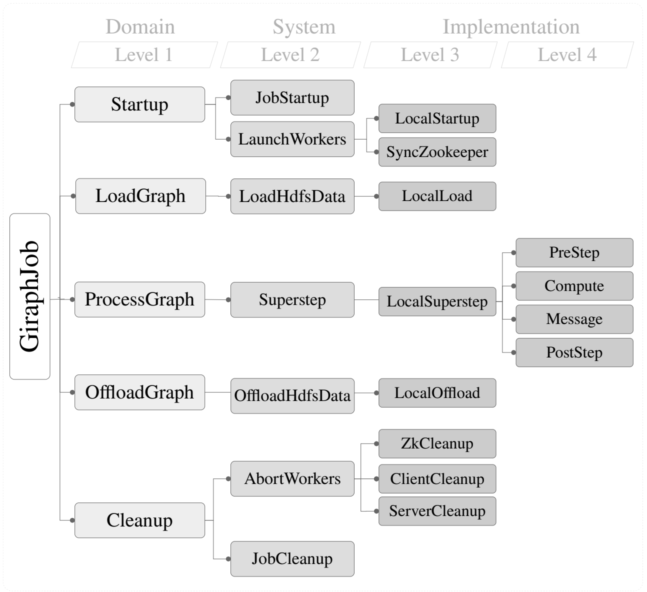

Figure 4: A *Granula* performance-model of Giraph.

To exemplify the results, Figure 4 depicts a 4-level Giraph model; finer-grained models are possible if needed.

At the domain level, a Giraph job, similar to other platforms specialized in graph processing, contains five common operations (see Section 3): Startup, LoadGraph, ProcessGraph, OffloadGraph, and Cleanup. Identical domain-level operations allow us to derive common performance metrics across all platforms, enabling cross-platform performance comparison and benchmarking [18]. For example, we can compare the performance of graph-processing platforms by deriving the processing time (T_p) from the duration of ProcessGraph, which for Giraph is the aggregated runtime of all Giraph-supersteps. Similarly, we can compare the input/output performance, by defining the I/O time (T_d) as the sum of the LoadGraph and OffloadGraph duration.

At the system level, each platform executes jobs through its specialized workflow. For example, during the Startup operation, Giraph initiates its workers by launching Yarn containers, whereas Powergraph uses MPI commands for deployment; during the LoadGraph operation, Giraph loads its data partitions from HDFS, whereas Powergraph reads directly from a local/shared file system. Although both Giraph and Powergraph apply iterative graph processing (a series of supersteps), they have significantly different implementation details.

At the implementation level, the system-level operations of Giraph can be further broken down into smaller operations. However, a full performance breakdown is time-consuming, and should only be done when analysts encounter actual performance issues. For example, to understand the performance overhead in algorithm execution, the Superstep operation can be further broken down into three stages, PreStep, Compute, and PostStep, which could then be further broken down into finer-grained operations.

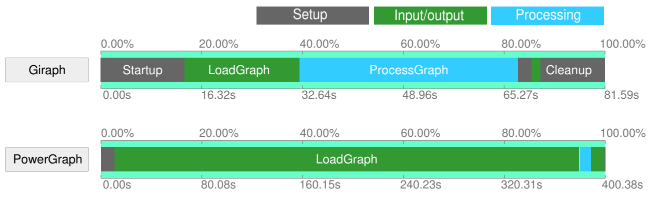

Figure 5: Job decomposition at the domain level.

4.2 Quantifying System Performance

To quantify system performance, *Granula* gathers empirical performance data by executing real-world graph processing jobs on these platforms. The collected data are automatically filtered, analyzed, and eventually stored in a performance archive, based on the *Granula* performance model defined by the analyst.

Figure 5 depicts the domain-level job decomposition of executing the *BFS* algorithm on *dg1000* graph using Giraph and Powergraph. Conceptually, the performance models at the domain level are similar across platforms (See Figure 3), but lead to significant performance differences during real-world execution. For Giraph, the setup (Startup and Cleanup), input/output (LoadGraph and OffloadGraph), and processing (ProcessGraph) operations contribute 30.9% , 43.3%, and 25.8% to the overall runtime, respectively. For Powergraph, despite a faster processing time, 94.8% of the runtime is spent on input/output operations (LoadGraph and OffloadGraph), leaving the algorithm execution (ProcessGraph) at under 3.1% of the runtime. The prolonged I/O time indicates performance issues in the data loading process of Powergraph.

Based on the domain-level job decomposition, analysts can identify potential performance issues in different stages of the system operation workflow, and make strategic decisions whether it is necessary to continue more fine-grained investigation to uncover the root cause of the performance issues.

4.3 Monitoring Resource Usage

To further help diagnose the problem, *Granula* can also make use of fine-grained performance data (such as the resource utilization, i.e., CPU usage) from the underlying cluster environment, and map these data to the each corresponding system operation. For example, Figures 6-7 depict the cumulative CPU usage of distributed Linux processes on eight DAS5 compute nodes mapped to Giraph and Powergraph jobs, respectively.

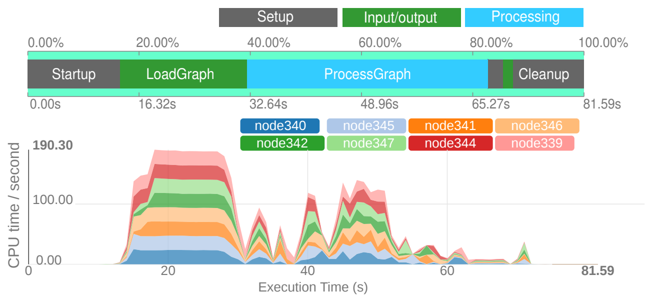

Figure 6: CPU utilization of Giraph operations.

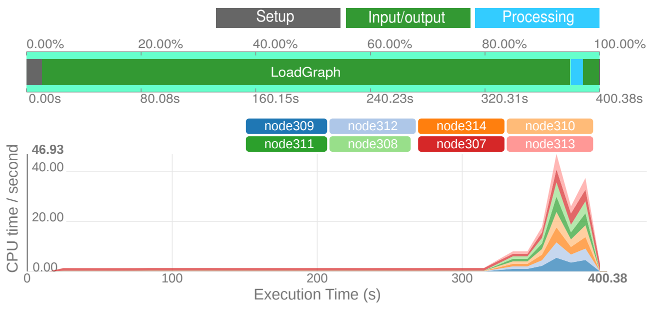

Figure 7: CPU utilization of Powergraph operations.

For Giraph (Figure 6), we observe that setup operations (Startup and Cleanup) are not compute-intensive. Surprisingly, the input/output operations (LoadGraph) are the ones that make heavy use of the CPU. There are also several peaks in CPU utilization during the algorithm execution (ProcessGraph), but in general the CPU resources are under-utilized. The low CPU utilization of the setup operations indicates that the performance bottleneck is not CPU-bound, but rather possibly caused by latency issues in the Giraph deployment mechanism. The high CPU utilization of I/O operations indicates a compute-intensive data loading mechanism, which shows a clear direction for tuning or optimization. During the algorithm execution, the peaks of and differences in CPU usage per node indicate workload-imbalance between supersteps, and also between compute nodes in the same superstep.

For Powergraph (Figure 7), we observe that, during the data loading stage (LoadGraph), only one compute node is utilizing the CPU, while the other nodes are idle. This confirms that only one compute node is responsible for loading the graph dataset from the local/shared file system to memory. Only towards the end of data loading stage (LoadGraph), other nodes start to participate in constructing the in-memory graph data structure and continue with executing the algorithm (ProcessGraph). This indicates that the data loading mechanism of Powergraph, which loads input sequentially from the storage system, is not a good fit for the distributed execution environment.

The mapping of fine-grained performance data to corresponding system operations allows analysts not only to pin-point hidden performance issues in system operations, but also to hypothesize about the types of possible performance issues in the different stages and to test these hypotheses.

4.4 Visualizing System Behavior

To understand better how finer-grained operations function, analysts can also use *Granula* to visualize operations at the system and, especially, at the implementation level. For example, Figure 8 depicts the details in the execution of superstep, when running the *BFS* algorithm on Giraph.

As defined by the Giraph performance model (See Figure 1), each Superstep operation can be further decomposed into PreStep, Compute, and PostStep. We observe that the compute workload is not distributed evenly among supersteps; for example, superstep Compute-4 takes significantly longer to run than the others. We also observe that the workload is not balanced among workers; for

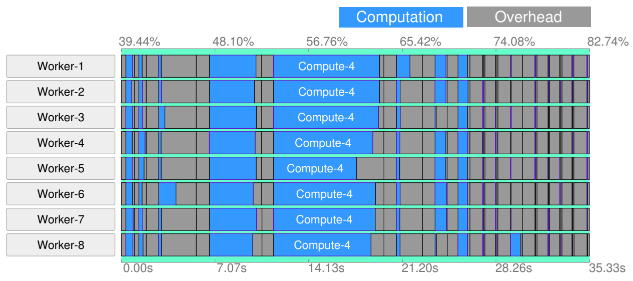

Figure 8: Compute-workload distribution among workers, as visualized by *Granula*.

example, for Compute-4 and for other supersteps, some workers take more time to complete their computation than others, while the other Giraph workers are simply waiting at the superstep barrier. Furthermore, there are significant performance overheads in superstep synchronization, which can be observed in the idling time, that is, large (visible) gray PreStep and PostStep operations around the light-blue Compute operations.

The visualization of workload imbalance in the Giraph example indicates that performance analysts can use *Granula* to test existing performance issues, especially of the kind visible only at very fine granularity.

5 RELATED WORK

Overall, *Granula* adds to related work in Big Data support for a standardized and automatable evaluation process. It also improves the specific step of performance evaluation: performance studies, including our previous work, either employ an inaccurate coarse-grained evaluation method that is typical to benchmarking [8, 14, 15, 20, 23]; or focus on a reduced subset of internal operations of Big data platforms, e.g., use of fine-grained hardware resources [11, 22]. In contrast, through its incremental evaluation, *Granula* empowers the analyst to control the trade-off coarse-/fine-grained analysis.

Our work on *Granula* also extends significantly our state-of-the-art work in benchmarking graph-processing platforms [18]—with *Granula*, we focus on a complete evalution process including modeling, and on fine-grained analysis vs. coarse-grained benchmarking.

6 CONCLUSION AND FUTURE WORK

End-to-end, shareable performance evaluation remains a needed yet missing capability in Big Data processing. In this work, we propose the design of *Granula*: a fine-grained performance analysis system for Big Data platforms which facilitates performance modeling, monitoring, archiving, and visualization. *Granula* applies an incremental evaluation process, in which analysts can develop their understanding of a Big Data platform by systematically building a comprehensive performance model for that platform. This gives analysts a tool to control the trade-off between the high-accuracy of fine-grained analysis and the low-cost of coarse-grained analysis.

We demonstrate the capabilities of *Granula* by conducting fine-grained performance evaluation on two graph processing platforms, Giraph and Powergraph. The results show that our system can simplify the work of performance analysts, help them quantify the performance characteristics of system operations, identify hidden performance issues, and explain system behaviours with fine-grained performance data.

In our ongoing work, we continue toward our vision of *Granula*: to continue the development of our research prototype and fully implement the four main *Granula* modules to support an end-to-end performance evaluation process; to further enhance *Granula*'s ability to support performance analysis, for example on choke-point analysis and failure diagnosis; to help integrate performance analysis as part of standard software engineering practices, in the form of performance regression tests; to develop a larger library of comprehensive performance models for various types of large-scale graph processing platforms, and to conduct fine-grained performance analysis on more Big Data platforms.

ACKNOWLEDGMENTS

This research is supported by the Dutch NWO projects Vidi MagnaData, KIEM KIESA, COMMIT COMMIssioner; and by LDBC (ldbcouncil.org), originally funded by EU FP7-317548. This work is also partially supported by a donation and valuable input from Oracle Research Labs; we particularly thank Hassan Chafi, Sungpack Hong, and Davide Bartolini.

REFERENCES

[1] M. Anderson et al. GraphPad: optimized graph primitives for parallel and distr. platforms. In *IPDPS*, 2016.
[2] R. Angles et al. The linked data benchmark council: A graph and rdf industry benchmarking effort. *SIGMOD Rec.*, 43(1):27–31, May 2014.
[3] H. Bal et al. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer*, 49(5):54–63, 2016.
[4] D. Böhme et al. Caliper: performance introspection for HPC software stacks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016, Salt Lake City, UT, USA, November 13-18, 2016*, pages 550–560, 2016.
[5] D. Borthakur et al. HDFS architecture guide. *Hadoop Apache Project*, 53, 2008.
[6] A. Ching et al. One trillion edges: Graph processing at Facebook-scale. *PVLDB*, 8(12):1804–1815, 2015.
[7] N. Doekemeijer et al. A survey of parallel graph processing frameworks. 2014.
[8] B. Elser et al. An evaluation study of bigdata frameworks for graph processing. In *Big Data*, pages 60–67, 2013.
[9] O. Erling et al. The LDBC Social Network Benchmark: Interactive workload. In *SIGMOD*, pages 619–630, 2015.
[10] European Commission. A digital single market for Europe: Commission sets out 16 initiatives to make it happen. http://europa.eu/rapid/press-release_IP-15-4919_en.htm, 2015.
[11] M. Ferdman et al. Quantifying the mismatch between emerging scale-out applications and modern processors. *ACM Trans. Comput. Syst.*, 30(4):15:1–15:24, 2012.
[12] A. Gharaibeh et al. A yoke of oxen and a thousand chickens for heavy lifting graph processing. In *PACT*, pages 345–354. ACM, 2012.
[13] J. E. Gonzalez et al. PowerGraph: Distributed graph parallel computation on natural graphs. In *OSDI*, pages 17–30, 2012.
[14] Y. Guo et al. How well do graph-processing platforms perform? an empirical performance evaluation and analysis. IPDPS, 2014.
[15] M. Han et al. An experimental comparison of pregel-like graph processing systems. *PVLDB*, 7(12):1047–1058, 2014.
[16] S. Hong et al. PGX.D: a fast distributed graph processing engine. In *SC*, pages 58:1–58:12, 2015.
[17] IDC. The digital universe of opportunities. https://www.emc.com/collateral/analyst-reports/idc-digital-universe-2014.pdf, 2014.
[18] A. Iosup et al. LDBC Graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms. *PVLDB*, 9(13):1317–1328, 2016.
[19] Y. Low et al. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *PVLDB*, 5(8):716–727, 2012.
[20] Y. Lu et al. Large-scale distributed graph computing systems: An experimental evaluation. *PVLDB*, 8(3), 2014.
[21] G. Malewicz et al. Pregel: a system for large-scale graph processing. In *SIGMOD*, pages 135–146, 2010.
[22] L. Nai et al. GraphBIG: understanding graph computing in the context of industrial solutions. In *SC*, pages 69:1–69:12, 2015.
[23] N. Satish et al. Navigating the maze of graph analytics frameworks using massive datasets. In *SIGMOD*, pages 979–990, 2014.
[24] N. Sundaram et al. Graphmat: High performance graph analytics made productive. *PVLDB*, 8(11):1214–1225, 2015.
[25] V. Vavilapalli et al. Apache Hadoop YARN: Yet another resource negotiator. In *SOCC*, page 5, 2013.
[26] A.-J. Yzelman et al. High-level strategies for parallel shared-memory sparse matrix-vector multiplication. *TPDS*, 25(1):116–125, 2014.