Universidade Federal do Rio Grande do Sul
Instituto de Informática
Grupo de Processamento Paralelo e Distribuído

# Proceedings

## XV Workshop de Processamento Paralelo e Distribuído

WSPPD 2017

4 de Setembro de 2017
Auditorio Centro de Eventos do Instituto de Informática
Porto Alegre, RS
ISSN: 2175-6848



# List of Sessions

# Energy Consumption and Performance Analysis Between SSD and HDD *

Pablo J. Pavan [1], Vinícius R. Machado [2], Jean L. Bez[2],
Edson L. Padoin[1,2], Francieli Z. Boito[2], Philippe O. A. Navaux[2], Jean-François Méhaut[3]

[1] Universidade Reg. do Noroeste do Estado do Rio G. do Sul (UNIJUI) — Ijui, RS, Brazil
[2] Universidade Federal do Rio Grande do Sul (UFRGS) — Porto Alegre, RS, Brazil
[3] Université Grenoble Alpes – Grenoble, France

{pablo.pavan, ricardo.lorenzoni, padoin}@unijui.edu.br,
{jean.bez, francieli.zanon, navaux}@inf.ufrgs.br, jean-francois.mehaut@imag.fr

## Abstract

*This paper presents an energy efficiency and I/O performance analysis of low-power architectures when compared to conventional architectures. Despite the fact, the power demand of the storage device amounts for a small fraction of the power demanded by whole system, significant increases are observed when accessing the storage device. We investigate the access pattern impact on power demand, looking at the whole system and at the storage device by itself. All tested configurations are compared regarding energy efficiency. We show the viability of replacing conventional servers by low-power alternatives depends on the characteristics of the expected workload.*

## 1. Introduction

The increase in processing power of High-Performance Computing (HPC) architectures had, as a consequence, the growth in power demand. Taking this matter into consideration, a DARPA report suggests a limit of 20 MW of power demand for future HPC systems, which are expected to achieve exaflops of performance [2].

This premise has led HPC researchers to seek alternatives that respect the given power limit. A strategy could be the use of low-power architectures, replacing traditional processors by Advanced RISC Machine (ARM) processors. Despite presenting lower performance, these ARM processors provide better energy efficiency for some scientific applications [5].

In any typical computational system, the processor is not the solely responsible for the power demand of the system. Because of the increasing difference between processing and data access speeds, it is common for applications to spend a significant share of their runtime reading or writing data. Therefore, it is imperative that the energy consumption of the system, during these input/output (I/O) operations, be investigated and improved.

In this paper, we conduct a comparative study of I/O performance and power demand. We analyze the power demand of the storage devices and the system as a whole to better correlate it with the measured bandwidth. We compare a machine with a traditional processor to a Multi-Processor System-on-Chip (MPSoC), with a low-power ARM processor. Our analysis considers Hard Disk Drives (HDDs) and Solid State Drives (SSDs) as storage devices. Our goal is to determine the I/O workloads that benefit most in using such low-power architectures as data servers.

The remainder of this paper is organized as follows. Section 2 discusses related work. The experimental environments and methodology are detailed in Section 3. Results and discussed are presented in Section 4. Finally, Section 5 concludes this paper and discusses possible future work.

## 2. Related Work

Sawada *et al.* investigate the power consumption of a server to perform storage operations and computations. They defined a simple power consumption model of a server, abstracting parameters like the number of processes,

* This work was supported by CNPq, CAPES, FAPERGS and FINEP. This research has received funding from the European Community's Seventh Framework Programme (FP7-PEOPLE) under grant agreement number 295217, funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E Project, grant agreement number 689772 and STIC-AmSud/CAPES scientific-technological cooperation program under EnergySFE research project grant 99999.007556/2015-02.

which dominate the power consumption [6]. Hui *et al.* analyzes the performance and the energy consumption of some SSDs using different combinations of file systems and I/O blocks [1].

Nijim *et al.* combine flash-based storage devices (SSDs) with hard drives to provide storage with lower energy consumption. This is achieved by using the SSD as a cache for the hard drives [4]. This hybrid storage strategy is exploited by others to provide high performance for I/O servers [7]. In these cases, the SSD is used only as a cache because of its high cost per byte, which makes the total replacement of hard drives unfeasible.

In a previous work [3], an initial evaluation was done comparing low-power architectures with common architectures. This evaluation was done under the HPC point of view, aiming to analyze the workloads of scientific applications. The work suggested that MPSoCs were a good low-power alternative for read workloads. The research presented in this work is more complete — exploring more configurations and different workloads — and reaches different conclusions since workloads are more intensive.

## 3. Experimental Methodology

Two environments were used for the experiments discussed in this work. The first, named **PC**, is a traditional desktop, with an Intel Core-I7 4790 processor of 3.6 GHz base clock frequency. The equipment has 16 GB RAM operating at 1600 MHz.

The second environment is a MPSoC CubieTruck, with a SoC A20 manufactured by AllWinnerTech, and a dual GPU MALI400 MP2, called **MPSoC**. The processor is a Dual Core ARM Cortex-A7 at 920 MHz frequency. The equipment has 2 GB of RAM operating at 480 MHz frequency.

Four storage devices were used for the experiments, two SSDs and two HDDs selected to cover different characteristics. They are presented in Table 1.

| | Type | Manufacturer/Model | Capacity (GB) | Interface | RPM |
|---|---|---|---|---|---|
| **HDD1** | HDD | Seagate ST1000LM035-1RK172 | 1000 | SATA III | 5400 |
| **HDD2** | HDD | Seagate ST96023AS | 60 | SATA II | 7200 |
| **SSD1** | SSD | Samsung PM871 | 240 | SATA III | - |
| **SSD2** | SSD | Kingston SV300S37A120G | 120 | SATA III | - |

**Table 1. Storage devices used for the experiments**

The benchmark chosen for the tests was FIO, [1] selected for being widely used and for allowing the description of the desired access patterns. The **write** experiments were conducted in each configuration **with** and **without** the usage of the buffer cache. On the other hand, we do not include read experiments with the cache because in this situation the behavior depends on previous accesses. If the same data was accessed recently, it may still be in the cache, but a variety of situations may occur.

Regarding the type of operation and the spatiality, two access patterns were generated: sequential write, random write, sequencial read, random read. Additionally, two request sizes were evaluated: 32 KB and 4 MB. Each test was executed for 20 GB of data with an execution time limit of 60 seconds - the test stopped when the first of these two conditions was met. We use the bandwidth reported by FIO as the performance metric for these experiments.

Considering the eight configurations of equipment and storage device, two cache options for half the experiments (with or without it), four operations, and two request sizes, a total of 96 experiments are analyzed in this paper. Each one of them was repeated 10 times, and different experiments in the same equipment with the same storage device were executed in random order, to avoid unexpected effects. A minimum 20-seconds delay is guaranteed between tests, so the power demand of the environment stabilizes. Moreover, the "`sync`" command is used between tests that use the buffer cache to make sure they are independent.

To measure the power demand, we employed an Agilent oscilloscope model DSO6014A. This oscilloscope was connected via USB to a computer, where the BenchVue software logs captured data. A power tip model 1146A, manufactured by Agilent, was used to measure the current for the entire equipment. Current for the storage devices was measured from the Hall effect, with an Allegro solution model ACS712T connected to the oscilloscope. Instantaneous voltage and current measurements are obtained every 500 ms.

## 4. Results and Discussion

The results have shown that all devices suffer in performance when used in the MPSoC. In the PC, write performance was up to 1062% higher and read performance up to 522% higher. This difference was higher with SSDs than with HDDs. The power demand analysis has shown SSDs do not demand as much power in the MPSoC than in the PC, indicating the limitations imposed by other components keep the SSDs from reaching their peak performance when accessed by the MPSoC.

All tested scenarios were compared regarding energy efficiency, in bytes per Joule. Mainly because of access patterns impact on performance, impacts on energy efficiency were also observed. In some situations, using the cache led to up to 1737% higher energy efficiency, issuing large requests increased it in up to 1277%, and using sequential spatiality instead of random had an impact of up to 2836%.

1 `https://linux.die.net/man/1/fio`

Using SSDs leads to up to $6675\%$ higher energy efficiency than using HDDs. Moreover, when using SSDs energy efficiency is higher in the PC than in the MPSoC—up to $196\%$ for write workloads and $564\%$ for read workloads. This happened because of the large difference in performance observed with SSDs (higher in the PC). On the other hand, when using HDD2, since the device is not fast enough to be severely limited by the MPSoC infrastructure, the performance difference was not that large and hence using the MPSoC results in higher energy efficiency (up to $166\%$).

The results presented in this study indicate that replacing the traditional server by multiple low-power ones only results in higher energy efficiency if the PC uses HDDs for storage, and the MPSoC uses SSDs. In this case, we could observe $8\%$ lower power demand by replacing the PC by three MPSoC, with no harm to sequential write bandwidth and increasing random write bandwidth in up to $40\%$. Regarding read workloads, this replacement could be by $1.4$ MPSoC servers to keep the same sequential read bandwidth with small requests, increase sequential read bandwidth with large requests in up to $61\%$ and increase random read bandwidth in up to $294\%$.

The replacement of traditional servers by low-power ones also makes sense if both use HDDs for read workloads: $2.2$ MPSoC servers would be required in the scenario with HDD1, resulting in $20\%$ lower power demand, and $1.2$ MPSoC in the scenario with HDD2, demanding $42\%$ less power. These replacements would keep the same sequential read bandwidth, and increase the random read bandwidth in up to $120\%$. Nonetheless, write workloads would observe lower performance.

## 5. Conclusion and Future Work

Despite the fact, processing is responsible for most of the power demanded at computational systems, applications spend longer periods of time performing I/O, and thus the energy efficiency of these operations is also an important topic for HPC. In this paper, we conducted a comparative study of I/O performance and energy efficiency between a traditional computer and a low-power alternative, with HDDs and SSDs and under different types of workload.

Possible directions for future work include expanding the investigation to other processors and storage devices. Furthermore, we would like to consider the traditional and low-power alternatives as storage servers, receiving requests through the network and processing them.

## References


[1] S. Hui, Z. Rui, C. Jin, L. Lei, W. Fei, and X. C. Sheng. Analysis of the file system and block io scheduler for ssd in performance and energy consumption. In *2011 IEEE Asia-Pacific Services Computing Conference*, pages 48–55, Dec 2011.

[2] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, et al. Exascale computing study: Technology challenges in achieving exascale systems. Technical report, 2008.

[3] V. Machado, A. Braga, N. Rampon, J. Bez, F. Boito, R. Kassick, E. Padoin, J. Diaz, J.-F. Méhaut, and P. Navaux. Towards energy-efficient storage servers. In *Proceedings of the Symposium on Applied Computing*, SAC '17, pages 1554–1559, New York, NY, USA, 2017. ACM.

[4] M. Nijim, A. Manzanares, X. Ruan, and X. Qin. Hybud: An energy-efficient architecture for hybrid parallel disk systems. *Proceedings - International Conference on Computer Communications and Networks, ICCCN*, 0845257(2005), 2009.

[5] E. L. Padoin, L. L. Pilla, M. Castro, F. Z. Boito, P. O. A. Navaux, and J.-F. Mehaut. Performance/energy trade-off in scientific computing: The case of ARM big.LITTLE and Intel Sandy Bridge. *IET Computers & Digital Techniques*, 2(3):1–14, 2014.

[6] A. Sawada, H. Kataoka, D. Duolikun, T. Enokido, and M. Takizawa. Power consumption and computation models of a storage server. In *Broadband and Wireless Computing, Communication and Applications (BWCCA), 2015 10th International Conference on*, pages 472–477. IEEE, 2015.

[7] B. Welch and G. Noer. Optimizing a hybrid SSD/HDD HPC storage system based on file size distributions. In *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–12, 2013.

# TWINS: Server Access Coordination in the I/O Forwarding Layer

Jean Luca Bez
Informatics Institute
Federal University of Rio Grande do Sul
jean.bez@inf.ufrgs.br

Francieli Zanon Boito
Department of Informatics and Statistics
Federal University of Santa Catarina
francieli.boito@posgrad.ufsc.br

Lucas M. Schnorr
Informatics Institute
Federal University of Rio Grande do Sul
schnorr@inf.ufrgs.br

Philippe O. A. Navaux
Informatics Institute
Federal University of Rio Grande do Sul
navaux@inf.ufrgs.br

Jean-Francois Mehaut
Universite de Grenoble Alpes
INRIA
CEA LIG Laboratory
Grenoble, France
jean-francois.mehaut@imag.fr

## Abstract

*This paper presents a study of I/O scheduling techniques applied to the I/O forwarding layer. In high-performance computing environments, applications rely on parallel file systems (PFS) to obtain good I/O performance even when handling large amounts of data. To alleviate the concurrency caused by thousands of nodes accessing a significantly smaller number of PFS servers, intermediate I/O nodes are typically applied between processing nodes and the file system. Each intermediate node forwards requests from multiple clients to the system, a setup which gives this component the opportunity to perform optimizations like I/O scheduling. We evaluate scheduling techniques that improve spatiality and request size of the access patterns. We show they are only partially effective because the access pattern is not the main factor for read performance in the I/O forwarding layer. A new scheduling algorithm, TWINS, is presented to coordinate the access of intermediate I/O nodes to the data servers. Our proposal decreases concurrency at the data servers, a factor previously proven to negatively affect performance. The proposed algorithm is able to improve read performance from shared files by up to 28% over other scheduling algorithms and by up to 50% over not forwarding I/O.*

# Detecting Memory-Bound Parallel Regions to Improve the Energy-Efficiency of Applications

Gabriel Bronzatti Moro*, Lucas Mello Schnorr*
* Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil

***Abstract*—Performance and energy consumption are fundamental requirements in some areas. It is a challenge to maintain the same performance using less energy. In this article, we analyze the behavior of the Lulesh application using a pre-run that provides the necessary information to identify its memory-bound regions. Our results present the detection of the Memory-Bound behavior of such application. Among these results, it is possible to visualize each region covered by the thread and its respective location in the source code of the application, as well as its misses rate using hardware counters provided by the PAPI tool. As future work, we intend to implement a library that will use this knowledge base to reduce the processor frequency in the Memory-Bound regions. Our assumption is that in these regions it will be possible to reduce the energy consumption of the application.**

## I. Introduction

Energy consumption is an important issue in large data centers as well as in battery-dependent devices. There is a trade-off between performance and energy consumption, enabling one to find a best configuration. Scientific applications, image and video processing are widely used in High-Performance Computing, they can pottentially be considered as Memory-Bound applications because they are limited by access time to memory [1].

DVFS (Dynamic Voltage and Frequency Scaling) approach is used to reduce the power consumption in Memory-Bound Applications, using a low processor frequency when the application is waiting for memory. Some studies investigate the appropriated frequency adjustment using information about applications behavior, platform characteristics, and load balancing. Freeh et al. [2] define an approach for distributed applications, Laurenzano et al. [3] present a technique for sequential and parallel applications, Spiliopoulos et al. [4] create a tool for sequential applications (using the loop level) and Millani and Schnorr [5] use a granularity by parallel regions for the frequency adjustment. Among those studies, it is possible to understand that for OpenMP parallel applications, it is necessary more efforts to optimize Memory-Bound regions in parallel applications using information about the applications behavior. In this context, our main objective is to identify precisely the Memory-Bound regions of an application, so that it is possible as future work to create an automatic approach with the variation of processor frequency by parallel region.

This paper is organized as follows. Section II shows some concepts about Application Behavior and Energy Consumption. In Section III, the related works regarding automatic phase detection for HPC applications. After it, the Section IV is presented our proposal and its corresponding methodology. The Section V describes the results. Section VI concludes the paper with the main contributions and future works.

## II. Application Behavior and Energy Consumption

Mapping, Stencil, and Reduction are parallel application models widely used for shared memory programming. Mapping uses a process that launches several threads (fork), after this division, a specific processing occurs in each thread or a similar processing on different data [6]. Stencil applications perform the convolution approach to obtain a new result. In this case, it is used "n" iterations to get the result for a particular matrix's element, using its neighboring cells [7]. Reduce (Division-and-Conquer) applications have a combination of small solutions obtained from the processing of parts of a collection of input elements. The output of this type of implementation is an unique result, which was calculated by consecutive reduction steps.

For example, a Vector Multiplication Algorithm is different than a Search Algorithm in Graphs. The Algorithm of Search in Graphs has a more Memory-Bound behavior than the Vector Multiplication Algorithm because when a search is performed on graphs, the nodes are far apart from each other whereas in Vector Multiplication Algorithm the elements are better located, allowing good Cache spatial location.

In Memory-Bound applications, the performance increases when the rate of misses in the L2 Cache is reduced [8]. Otherwise, Non-Memory-Bound applications, the performance does not improve by reducing cache misses rate. Hardware counters are used to trace such metrics, enabling one to collect the number of accesses to different Cache Levels, Main Memory, the number of instructions executed, and other data about the platform and application performance behavior.

In addition to the misses rate for the Cache L2 to define the behavior of an application, it is possible also use the Instructions by Cycle (IPC) metric. This metric allows to check how many instructions the program performs per processor cycle. That way, it enables to verify if reducing the number of instructions of an application it will occur the increase of its performance. If this happens it is possible to classify the application such as Non-Memory-Bound or CPU-Bound application [8].

The power consumption can be measured by energy sensors or estimated from models (theoretical estimate) [9]. Some tools

provide the access to energy sensors. Intel PCM is an example of a tool that allows obtaining the total energy consumption spent by the application, it provides the CPU (with Cache consumption), and Main Memory energy [10].

Several works use Dynamic Voltage and Frequency Scaling (DVFS) to save energy by reducing the processor frequency. This approach allows reducing the processor frequency in certain parts of the program execution, which have a more Memory-Bound behavior [11]. The CPUfreq framework allows changing the frequency used by the processor, this framework provides Performance, Powersave, OnDemand, Userspace, and Consertative mode [12]. Each of the modes defines a priority to be achieved. Performance uses the highest frequency available, unlike the Powersave mode that always use the lowest frequency to obtain the lowest possible energy consumption [13]. Still, OnDemand mode performs the frequency adjustment according to processor usage. Generally, CPUFreq is used with Userspace mode to adjust appropriately the processor frequency.

## III. Related Work

There is no definitive solution to detect if a code region is more memory or CPU-bound. It is possible to find investigations about phase detection to sequential [4], [3], distributed [2] and shared-memory parallel applications [3], [5].

Spiliopoulos et al. [4] show a tool called Power-Sleuth that is able to provide a detailed description of the behavior of an application when executed at a respective frequency. They use three techniques in their work: phase detection, Dynamic Voltage and Frequency Scaling (DVFS), and correlation models. Their approach identifies the application's regions using ScarPhase library that uses an execution history. The program's functions have a similar behavior defined by misses rate, number of access to memory, and others metrics. This article uses only sequential applications, the investigation of the Memory-Bound regions can be obtained in a coarser granularity in the timestamps between samples. On the other hand, parallel applications are executed on different threads, each thread may have a different behavior according to load balancing, application model, or communication type [4]. In addition to Spiliopoulos et al. [4] approach, Poellabauer et al. [1] describe a technique called Feed-back loop, this approach uses the metric MAR (data cache misses divided by instructions executed). From the MAR equation, it is possible to analyze the percentage of misses in the instructions executed by the application as a whole. This metric allows investigating the application's behavior in a particular architecture. The results show an energy saving of up to 27% for the six applications executed.

Laurenzano et al. [3] define an automated approach that allows selecting the most appropriate processor frequency for a given program loop. The processor frequency is chosen based on a static analysis (performed before execution) and another analysis performed during the execution time of the application, using the obtained traces. The authors use several benchmarks, based on the Pcubed (PMaC's Performance and Power benchmark) framework, which allows the exploration of different behaviors of interaction loops, in order to define a characterization for the target platform. Characterization of the platform defines values such as power consumption, performance, execution patterns and processor frequencies. The results obtained in the experiment can be used later as a knowledge base, so it is possible to visualize the behavior of the energy consumption when adjusting the characterization factors of the platform. Among the results obtained by the work, the best is the reduction of up to 10.6% in energy consumption

Unlike Laurenzano et al. [3], Freeh et al.[2] present an approach to distributed memory for MPI applications. This approach uses a frequency for each node, the frequency is defined by a heuristic called `Gear` that defines the gain between energy consumption and performance. With the trace obtained from a pre-execution, the approach defines blocks (like Basic Blocks) that perform operations. For each block the desired gain is set. The gain is the best configuration found between power consumption and performance for a certain phase of the application. Their results show an advantage for more than half of the applications performed. The best result obtained is the reduction of energy consumption by 9% and the execution time by 1%. On the other hand, the technique of Freeh et al. [2], Ge et al. [14] present an approach that also uses the DVFS technique for parallel applications in clusters, but in this approach rather than using the more Memory-Bound regions of the application to apply the frequency reduction, the authors perform the reduction of processor frequency when CPU performance is not required, for example when it occurs communication between MPI processes.

For parallel applications that use the OpenMP library, Millani and Schnorr [5] present an approach which analyzes the parallel regions of a program using a detailed analysis with the technique of Design of Experiments and Screening Design. The authors use seven benchmarks. Through the executions they conclude that it is possible to achieve considerable energy and performance gains from the use of their approach, depending on the behavioral characteristics of the application. The technique consists in manual code instrumentation to trace the parallel regions in the source code. Otherwise, the focus of our work is on the automatic identification of these parallel regions, based on specific hardware counters values to find L2 misses rate and IPC by parallel region. Our approach use L2 misses rate because it allows for compiled results that accentuate the application's most Memory-Bound points, different from the L3 misses rate.

## IV. Methodology

Our approach allows investigating the program's behavior using Score-p instrumentation to collect the hardware counters using PAPI (Performance Application Programming Interface). We focus on OpenMP parallel programs. Figure 1 shows all the steps of our methodology.

The first step is to use the Score-p tool (with the OTF2 trace file format) to trace the hardware counters in a per-

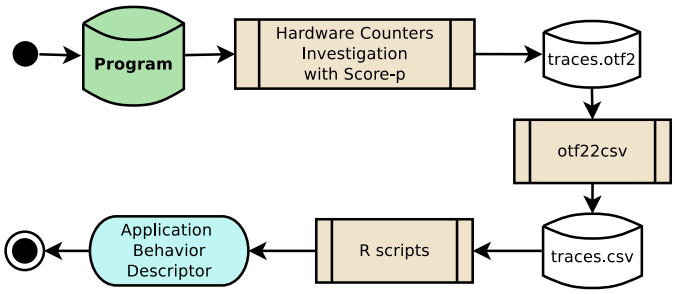



Fig. 1. The main steps of our methodology.

region fashion, where a region may be a parallel region with begin and end timestamps and the set of HW counters per thread. The hardware counters are those provided by PAPI tool, but we use only four hardware counters: PAPI_L2_TCA (Level 2 total cache accesses), PAPI_L2_TCM (Level 2 cache misses), PAPI_TOT_CYC (total cycles), and PAPI_TOT_INS (instructions completed). From this, it is possible to calculate the IPC (PAPI_TOT_INS divided by PAPI_TOT_CYC) and L2 misses rate (PAPI_L2_TCM divided by PAPI_L2_TCA).

The Akypuera `otf22csv` tool is used in a second step to convert the OFT2 file generated by execution with Score-p instrumentation to a CSV format. Because we use the Score-P tracing features, and not the profilling part, the granularity of the trace is a very detailed yet voluminous trace data per-thread and per-code region, enriched with HW counters metrics. To use the CSV information it is necessary to perform the grouping of all the metrics: thread, code region, timestamp and value of hardware counter. This grouping was performed using some statistical libraries of the R language. The CSV generated by the script accurately informs the behavior of the application, so it is possible to understand the most Memory-Bound points of the application, as well as Non-Memory-Bound.

From these previous steps, we intend to implement a library that uses the POMP2 library (from Opari2), so that this library automatically applies in Memory-Bound regions the use of an appropriated processor frequency according to previous steps. This library reads a configuration file, created by the analyst, to set the frequency appropriated to each code region in runtime.

In this work, we employ such methodology using the Lulesh application (using the params "-s 15 -i 100"). In the experiment we executed twice, the first to collect the amount of access and misses to l2 cache memory, the second to collect the number of CPU cycles and the total instructions. We use two executions to avoid the hardware counters saturation, so the first execution allows calculating the L2 misses rate and the other the calculate IPC in each region by thread. The execution platform used was the hype2, a Workstation with 2 processors Intel(R) Xeon(R) CPU E5-2650 2.30GHz, with 40 physical cores in total, and 126G of RAM memory.

## V. Results

The most important investigation is not the timestamp where the misses occur, but the code region that occurs the highest misses rate. In Figure 2 shows a subset of the parallel regions executed along time, in a case with 28 threads, for a very small time frame of ≈600 microseconds.

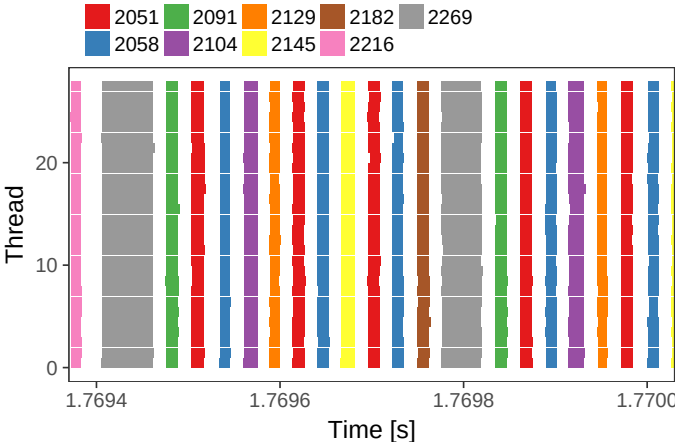



Fig. 2. Lulesh Behavior with a subset of nine parallel regions, identified by their starting line number in the source code.

Figure 2 shows nine regions (colors) per thread (on Y) as a function of time (on the X axis). These regions are identified by the starting line numbers that the tracer got from the unmodified source code of the application. We depict the behavior of 28 threads (represented by each row) and a temporal interval from 1.7694s to 1.7700s, just to show the richness of information we are able to collect through Score-P. One region might be executed multiple times depending on the application, as it is the case here for region identified by `2051` (the red color), which has been executed five times in this time frame.

For each of the 28 parallel regions of Lulesh, we evaluate their Memory-Bound behavior using per-region and per-thread HW-derived metrics. Figure 3 shows each of the regions (on X axis) for the two metrics: IPC (Instructions per Cycle, on top) and L2MR (L2 Miss Rate, on the bottom). These values are calculated by first taking the mean of the four measured metrics, then calculating the ratios that lead to IPC and L2MR. Only values for Thread 0 (the main thread) are considered, but all threads share the same HW counter behavior as we have observed. The highest L2MR is seem in regions `549` and `1171`, but only the latter has a low IPC. Further investigation is required to properly identify the true Memory-Bound regions, for example, by taking into account the average duration of the regions and eventually, other HW counters and derived metrics.

## VI. Conclusion

The main objective of this work is to analyze the behavior of the Lulesh application, analyzing all its parallel regions, specifically the regions that present Memory-Bound behavior. From the work it is possible to map each region, collecting hardware counters to calculate the L2MR (L2 Miss Rate) and the IPC (Instructions per Cycle) of each region during runtime.

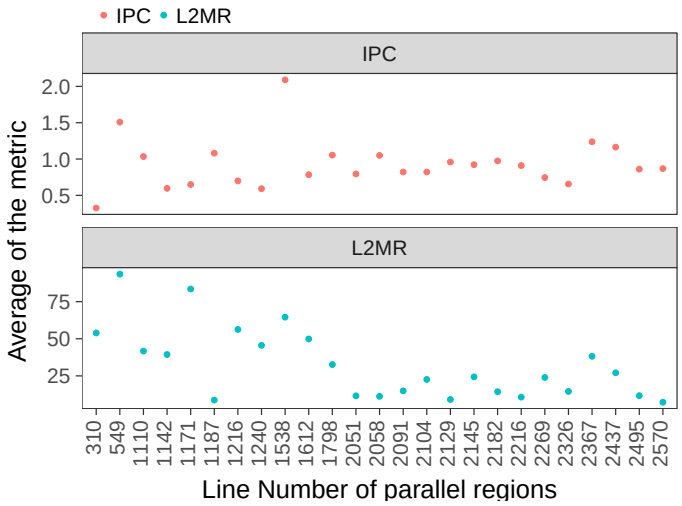



Fig. 3. IPC (Instructions per Cycle) and L2MR (L2 Miss Rate) for each of the 28 parallel regions of Lulesh that have been identified by Score-P.

The next step in this work is to implement a library that uses the results of the first analysis to properly set the appropriate processor frequencies in its Memory-Bound regions.

## VII. Acknowledgments

We thank FAPERGS/Inria ExaSE, FAPERGS GreenCloud, FAPERGS 16/2551-0000 354-8 (PPP 2014), CNPq 447311/2014-0, CNRS/LICIA Intl. Lab, the EU H2020 Programme and MCTI/RNP-Brazil under the HPC4E Project, grant 689772. We are also grateful for the masters scholarship and the HW resources provided by HPE under the HPCELO project. Some experiments were carried out at the Grid'5000 platform (https://www.grid5000.fr), with support from Inria, CNRS, RENATER and several other organizations.

## References

[1] C. Poellabauer, L. Singleton, and K. Schwan, "Feedback-Based Dynamic Voltage and Frequency Scaling for Memory-Bound Real-Time Applications," *In Proc. IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 234–243, 2005.

[2] V. W. Freeh, F. Pan, N. Kappiah, D. K. Lowenthal, and R. Springer, "Exploring the energy-time tradeoff in MPI programs on a power-scalable cluster," *Proceedings - 19th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2005*, vol. 2005, pp. 164–173, 2005.

[3] "Reducing energy usage with memory and computation-aware dynamic frequency scaling," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6852 LNCS, no. PART 1, pp. 79–90, 2011.

[4] V. Spiliopoulos, A. Sembrant, and S. Kaxiras, "Power-sleuth: A tool for investigating your program's power behavior," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*. IEEE, 2012, pp. 241–250.

[5] L. F. Millani and L. M. Schnorr, "Computation-aware dynamic frequency scaling: Parsimonious evaluation of the time-energy trade-off using design of experiments," in *European Conference on Parallel Processing*. Springer, 2016, pp. 583–595.

[6] P. Pacheco, *An introduction to parallel programming*. Elsevier, 2011.

[7] G. Roth, J. Mellor-crummey, K. Kennedy, and R. G. Brickner, "Compiling Stencils in High Performance Fortran," no. November, 1997.

[8] C. Jesshope and C. Egan, *Advances in Computer Systems Architecture: 11th Asia-Pacific Conference, ACSAC 2006, Shanghai, China, September 6-8*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006.

[9] A.-C. Orgerie, M. D. d. Assuncao, and L. Lefevre, "A survey on techniques for improving the energy efficiency of large-scale distributed systems," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 47, 2014.

[10] D. S. Silveira, G. B. Moro, E. H. da Cruz, P. O. Navaux, L. M. Schnorr, and S. Bampi, "Energy consumption estimation in parallel applications: an analysis in real and theoretical models," 2016.

[11] E. L. Seur and G. Heiser, "Dynamic Voltage and Frequency Scaling: the laws of diminishing returns," *Proceedings of the 2010 international conference on Power aware computing and systems*, pp. 1–8, 2010.

[12] T. I. Wiki, "DVFS User Guide," 2012. [Online]. Available: http://processors.wiki.ti.com/index.php/DVFS_User_Guide

[13] D. Brodowski and N. Golde, "CPU frequency and voltage scaling code in the Linux(TM) kernel," 2016. [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[14] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed dvfs scheduling for scientific applications on power-aware clusters," in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*. IEEE, 2005, pp. 34–34.

# Exploring the Impact of Workload Distribution in a Hybrid Edge and Cloud Application for Smart Grids

Otávio Carvalho, Manuel Garcia, Eduardo Roloff, Phillipe O. A. Navaux
Informatics Institute – Federal University of Rio Grande do Sul
Porto Alegre – Brazil
{omcarvalho,magnapa,eroloff,navaux}@inf.ufrgs.br

## Abstract

*Sensor networks have become ubiquitous through the advent of Internet of Things, spreading from smartphones to smart grids, generating increasing amounts of data in each time smaller time intervals.*

*However, the ability to aggregate and act upon such data gathered by sensors is still a significant research and industrial challenge. Devices that are able to collect and act on data at network edges are bounded by the amount of data that can be sent over networks.*

*In this paper, we analyse the impact of workload distribution in a smart grid application, evaluating how we can increase processing rates by leveraging each time more powerful edge node processors. Our results show that our test bed application is able to achieve processing rates of approximately 800k measurements per second using four edge node processors and a single cloud node.*

# Performance Variation of the Public Cloud

Guilherme G. Haetinger, Eduardo Roloff, Philippe O. A. Navaux
Informatics Institute
Federal University of Rio Grande do Sul - UFRGS
Porto Alegre, Brazil
{gghaetinger,eroloff,navaux}@inf.ufrgs.br

## Abstract

*Cloud Computing became a mainstream option as an environment to execute applications of any type. In recent years a lot of effort was made by the community to migrate High-Performance Computing (HPC) applications to the Cloud. However, the cloud still presents some drawbacks that need to be addressed. Aspects like network interconnection, raw performance, and high variability are the major barriers to the adoption of the cloud in large scale by the HPC community. In this work, we investigate the performance variation of Virtual Machines (VM) among different datacenters of the Microsoft Azure cloud provider. In our experiments, we observed that the same VM type presented a performance variation of 1.46%. Large performance variations are not desirable for HPC purposes, and this remains as an interesting research topic of cloud computing.*

## 1. Introduction

Cloud Computing public providers normally offer a large number of locations across the world. Research in High-Performance Computing (HPC) using the cloud normally focuses on performance evaluation [6, 7, 8] and application migration [1, 10, 2]. However, there is no guarantee from the provider side that an instance of the same type and size will present the same performance over all datacenter locations. Due to this, one important aspect that needs investigation is the performance variation of the same instance among different datacenters [9, 5].

In this work, we present a performance evaluation of the Microsoft Azure [1] cloud provider, using four different locations. We observed that, even using exactly the same instance type, there is a performance variation of up to 1.46%(at maximum measurement level) when comparing the best and worst datacenter.

1 https://azure.microsoft.com

The rest of this paper is organized as follow. The Section 2 provides a brief summary of the Microsoft Azure datacenters and motivates our work. In Section 3 we describe what our hypothesis is and how we verified it. Section 4 shows and explains our results so far. Finally, in Section 5 we discuss the future work and the next steps that will be made in our research.

## 2. Background

Microsoft Azure is a cloud platform that has datacenters all over the globe so that people can use them for storage and machine virtualization. They charge for what you use, in a monthly subscription, where price is relative to the machine's characteristics, having a price range from $13 - $2,000.

### 2.1. Instance Types

There are different machine specifications in their inventory. The main ones would be:

**A0-4 (Basic)**: 'A' Basic is an economically simple option for cloud development and storage.

**A8-11 (High Performance)**: A8-11 instances feature Intel® Xeon® E5 processors, used for parallel programming and, consequently, perfect for cluster workloads development and MPI execution.

**Av2 (Standard)**: Av2 Standard is the latest generation of A series virtual machines with similar CPU performance and faster disk, being more suitable for servers and web services.

**D2-64 (New Generation)**: D2-64 v3 instances are the latest generation of General Purpose Instances.

Azure has more machines types, a summary of instance variations of Azure can be found in Table 1.

| Designation | #Inst. | Price range |
|---|---|---|
| General Purpose | 27 | 0.018 – 3.200 |
| Compute Optimized | 5 | 0.060 – 0.997 |
| Memory Optimized | 20 | 0.148 – 8.690 |
| Storage Optimized | 4 | 0.344 – 2.752 |
| GPU/FPGA | 7 | 0.900 – 4.370 |
| HPC | 6 | 0.971 – 2.136 |

**Table 1. Instances characteristics of Microsoft Azure in West USA datacenters (verified on July 25, 2017). Price ranges are given in US$/h.**

### 2.2. Azure locations

Microsoft Azure has 26 datacenters active(2017) in 10 different countries that are available for regular users, as well as some datacenters that are suitable only for private usage (like the Department of Energy exclusive datacenter).

The locations where Azure is available are: Australia (2 sites), Brazil (1 site), Canada (2 sites), India (3 sites), USA (8 sites), Asia-Singapore(2 sites), Japan (2 sites), South Korea (2 sites), Europe (2 sites) and UK (2 sites).

## 3. Motivation and Proposal

Because of the large number of datacenters locations detailed in Section 2, we decided to research whether there are any differences between these datacenters' performances, since low performance variation is an important aspect for HPC [4]. In case we find any dissimilarity, we intend to discover why and how we can use that.

## 4. Evaluation

This section describes the methodology used in our experiments. The results are presented and described as well.

### 4.1. Methodology

To evaluate the datacenters' machines efficiency, we needed a benchmark to measure the processing power of each of the VM computer's core. For that reason we created a benchmark, using C programming language and OpenMP [3], to execute a test in parallel in every processing core. This benchmark, named Leveled Core Individual Efficiency Benchmark (LCIE), consists in generating random integers according to the load level selected for each core. The purpose of the LCIE benchmark is to simulate simply real-life applications that present a load imbalance and not constant mathematical algorithms. It was designed to simulate different loads using the available cores of the environment. The benchmark is still a work in progress, but the first prototype was used in this work.

To get the needed results for the research, we executed a benchmark with the given proportion of '100000' five times in a single VM in each of the following USA regions: West, East, South and North. We executed it in a A8 virtual machine with Linux Ubuntu 64-bits as the OS and these specifications:

- 8 cores;
- 56GB of RAM;
- 382GB of storage;

Each A8 machine has 8 cores, and we used our benchmark to simulate four load levels of an application. The loads that we simulated represent proportions of 100%, 75%, 50% and 25% in terms of application instructions. Each load level was executed by 2 cores in every execution so we could fit all cores in the benchmark.

### 4.2. Results

The Figure 1 shows the results of our experiments. We executed our benchmark with four different load levels, to simulate an application with load imbalance among its processes. Considering what was explained in subsection 4.1, the four levels simulate workloads of 25%, 50%, 75% and 100% and are represented in the $x$ axis. The execution time is represented in the $Y$ axis. The error level is plotted in the figure as well, and the variation was low, which means that our results are valid.

It is possible to see in the figure that we have performance differences in all levels. The best overall location was the East, performing better than all the other three location in all the load levels. The second best location, that is close to the East, was the South. The West and North locations presented less performance than East and South. One important thing that we want to remark is the 75% load level for the West and North locations. West was slightly faster than North in all other levels, but in the 75% level, the North surpasses the West. This means that we observed performance differences according with the usage of the cores.

The Table 2 shows the performance gains or losses of all the location for all the levels compared to the West location. As we can see, there is a variation among the levels and their order of efficiency is not constant at all, specially the 100% level, in which the improvements of the East and South locations in relation with the others presented a huge decrease of the performance gain and we could not determine a pattern. The North location presented a chaotic behavior, taking the worst efficiency place in most of the times.

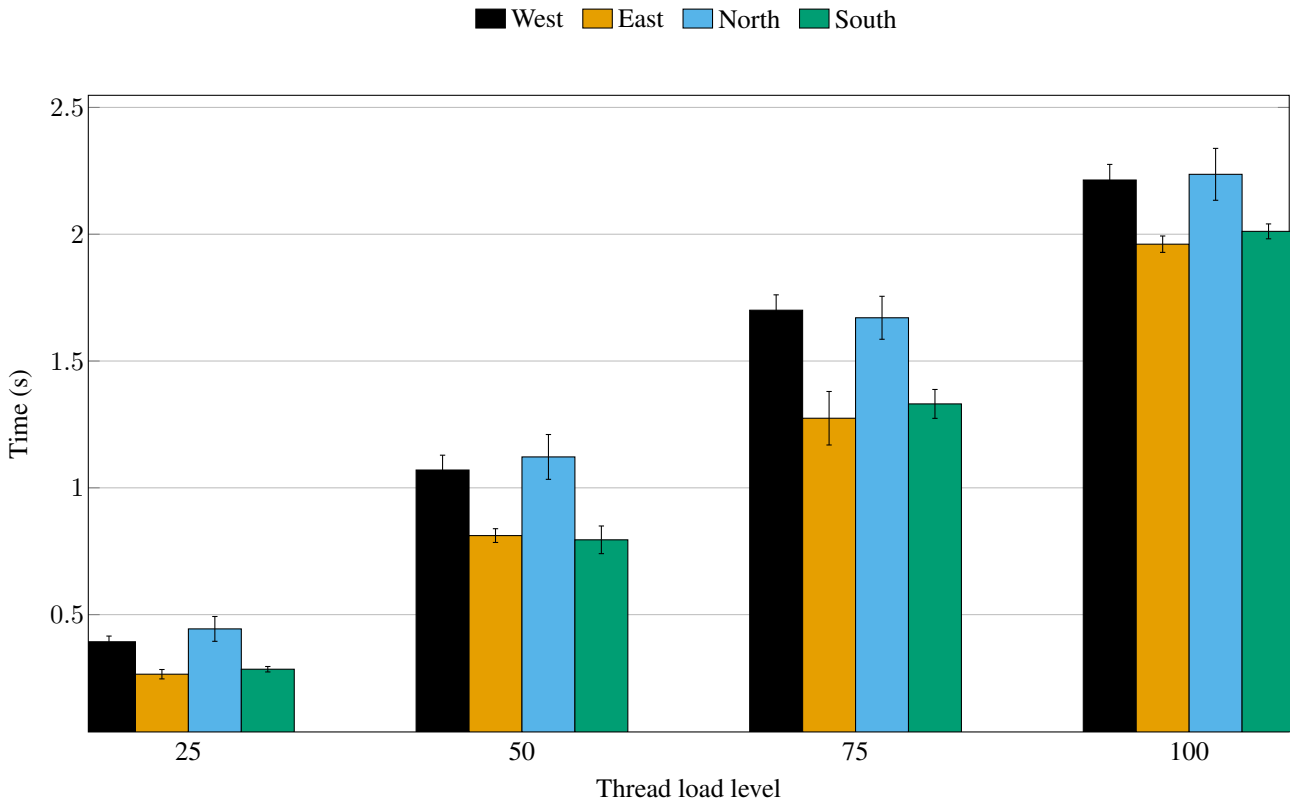



**Figure 1. Performance Results for the USA main regions**

As we gather the results in Figure 1, we can conclude that there is a significant difference between the instances from different locations. This performance variance is not desirable for HPC users, because it is important to have an stable environment.

The variance of data can affect the use from millions of public cloud users negatively by not providing the best experience possible on cloud workloads. However, both the differences of performance among these four locations and the differences observed in performance gains and losses for the different load levels are promising research topics that need further investigation.

| Load Level | East | North | South |
|---|---|---|---|
| 25 | 48.25% | -11.43% | 38.03% |
| 50 | 31.85% | -4.60% | 34.62% |
| 75 | 33.43% | 1.79% | 27.76% |
| 100 | 12.91% | -1.00% | 10.07% |

**Table 2. Performance gains and losses using the West location as baseline.**

## 5. Discussion

The Cloud Computing paradigm has a tremendous potential to be the *de-facto* standard environment for all kinds of applications. Moreover, it presents unique characteristics that are very promising for HPC users, such as the capability to build a completely customized environment in few minutes.

In this article, we focus on the performance differences among the same machine configuration used in different environments. Our results proved that there is a performance difference, even when we expected the same performance, because the configuration of the instances was the same.

We introduce our performance benchmark project as well. It will be used to test diverse cloud environments by simulating applications with processes that have different loads.

As future work, we will continue to investigate the cloud as an environment to execute HPC applications, with the main focus on the view from the user's side. We will create a mechanism for profiling the environments and help the user to choose the best suitable to their needs.

In the benchmark side, we will continue to improve it to create a complete toolkit to test the cloud. The purpose is to offer a tool that helps the user to measure the efficiency

of different cloud environments to help him or her to save money in their executions. As far as we know there is not a benchmark that was designed specifically for the cloud with focus in the HPC community.

**Acknowledgments.** This research received funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E project, grant agreement no. 689772. Additional funding was provided by FAPERGS in the context of the GreenCloud Project.

## References

[1] P. V. Beserra, A. Camara, R. Ximenes, A. B. Albuquerque, and N. C. Mendonça. Cloudstep: A step-by-step decision process to support legacy application migration to the cloud. In *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2012 IEEE 6th International Workshop on the*, pages 7–16. IEEE, 2012.

[2] E. D. Carreño, E. Roloff, and P. O. A. Navaux. Towards weather forecasting in the cloud. In *24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2016, Heraklion, Crete, Greece, February 17-19, 2016*, pages 659–663, 2016.

[3] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[4] Y. El-Khamra, H. Kim, S. Jha, and M. Parashar. Exploring the performance fluctuations of hpc workloads on clouds. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 383–387. IEEE, 2010.

[5] S. K. Garg, S. K. Gopalaiyengar, and R. Buyya. Sla-based resource provisioning for heterogeneous workloads in a virtualized cloud datacenter. In *International conference on Algorithms and architectures for parallel processing*, pages 371–384. Springer, 2011.

[6] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed systems*, 22(6):931–945, 2011.

[7] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. A performance analysis of ec2 cloud computing services for scientific computing. In *International Conference on Cloud Computing*, pages 115–131. Springer, 2009.

[8] E. Roloff, M. Diener, A. Carissimi, and P. O. Navaux. High performance computing in the cloud: Deployment, performance and cost efficiency. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 371–378. IEEE, 2012.

[9] A. Shieh, S. Kandula, A. G. Greenberg, and C. Kim. Seawall: Performance isolation for cloud datacenter networks. In *HotCloud*, 2010.

[10] V. Tran, J. Keung, A. Liu, and A. Fekete. Application migration to cloud: a taxonomy of critical factors. In *Proceedings of the 2nd international workshop on software engineering for cloud computing*, pages 22–28. ACM, 2011.

# Impacts of the Stream-Processing Model on Multi Geo-spatial Environments

Paulo Souza Junior, Alexandre Veith, Kassiano Matteussi, Claudio R. Geyer
Federal University of Rio Grande do Sul - Porto Alegre, Brazil
Institute of Informatics - PPGC
prrsjunior@inf.ufrgs.br

## Abstract

*Stream-Processing Systems (SPS) has become a crucial model in scientific and industrial domains in recent years. In this context, Multi Geo-spatial Environments are communally to host real time applications. It incurs in substantial communication costs, becoming a dominant operational expenditure factor. This work aims to evaluate the application slow-down impact in geographically distributed environments over different architecture models. The experiments were performed using a real scenario deployed on Microsoft Azure, where models are introduced and compared. Some of those models present up to 33% performance enhancement using batch-sized strategies.*

## 1. Introduction

The most different technologies are converging to exchange data, as a consequence Internet of Things (IoT) is taking a vital place on the Big Data Analytics. Googles MapReduce (MR) has introduced a simple and scalable data processing technique that deals with the massive amount of data generated by all the things. Since the time of its introduction, Big Data Analytics has attracted a good amount of attention from both industry and academia. According to IDG, in the period 2014-2015 there was an increase of 125% in the number of organizations that implemented or deployed data-driven projects. Also, IDG estimates that companies will spend approximately 13.8 million dollars per annum on them in the next few years. Moreover, 58% of business investments are related to Big Data Analytics. Most of the initial use cases were in batch analytics (i.e., Batch-Processing), which meant that data was stored on a disk and then periodically processed by walking through it. However, Batch-Processing was not structured to have low latencies in the processing of the batches. Thus, a new model was introduced - Stream-Processing.

In recent years, this subclass of Big Data called fast data (i.e., high-speed real-time and near-real-time data streams) has also witnessed a vast increase in volume and availability. These specific data, (often denoted as events), are usually characterized by a small unit size (in the order of kilobytes) but have overwhelming collection rates. As a result of this continuous flow of data a Stream-Processing approach has emerged (Apache Storm [12], Apache Spark [15], Apache Flink [1], S4 [10], Apache Samza [16], and so on).

According to [13], Stream-Processing have been applied on multi-geospatial (i.e. multi-cloud) environments and it has pulled off for scientific discovery. Stream-processing allows to perform most of the processing independently on the data partitions across different sites and then to aggregate the results in a final phase. In some of the largest scenarios, the data sets are already partitioned for storage across multiple sites, which simplifies the task of preparing and launching a geographical-distributed processing. These scenarios include exponentially expanding data, and according to IDC by 2020 there will be around 44 Zettabytes of data that will require processing. Multi-Cloud (MC) has characteristics as heterogeneity and variety of network bandwidth. The use of MC is inevitable since a single site will not be enough to handle the high volume of data. As pointed out by Smartinsights, since 2013 the growth of data generated per minute in the main companies has been exponential.

Therefore, this article intends to overcome environmental limitations, proposing a model that is followed by a technique. From this assumption we assume that problems will arise with this kind of environment as network latency and data placement causing slow down in applications. A technique is mandatory to improve the throughput of event streams between distributed environments.

To address this challenge the data clustering and a small batches strategy will be used to obtain low latency for the stream-based processing ([3] and [14]). Finally, the evaluation is conducted by a comparative study of batch sizes, looking for particular approaches to batch the streams (i.e., time-based, event-based or hybrid form). In contemplation of determine which is the more adequate technique for the Stream-Processing in this scenario.

## 2. Background

This work will seek to reach a geographically distributed environment that has its decoupled functions as shown in Figure 1. The components mentioned here are related to the role of computing resources. Figure 1 presents a Geographically Distributed decoupled Environment composed by a Data Source which represents the data producer, Message Broker that handles message queues, and also by a Stream-processing engine as the core for data analysis.

Still, by the adoption of this architecture model is possible to raise the resource processing pool (e.g., stream processing clusters) at any time. The structure will guarantee the fittable in and out of the resources in the network.

The adopted kind of architecture is due to the possibility to get over the elasticity problem. In this scenario, they have created different execution clusters and therefore allowing the insertion of computational resources at any time. The structure will fit the in and out of the resources in the network.

Moreover, the distributed infrastructure for such processing model should not have restrictions on some online resources (i.e., environments should automatically adapt to

the requests, scalability). The growth of the Internet of Things (IoT) supports that point, where more and more sensors and "things" that are data sources are present in our applications.

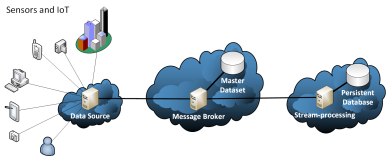



**Figure 1. Geographically Distributed decoupled Environment**

## 2.1. Multi Geo-Spatial Environments

The volume and velocity of data generation have been evolved in these last years. Consequently, it leads to the physical capacity constraint problems (i.e., single data center - Figure 2) [6].

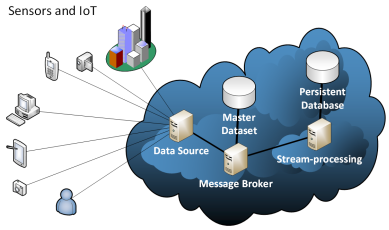



**Figure 2. Single Data Center**

Such restrictions related with Single Data Center can be seen as follows:

- **Memory**: Applications in Stream-processing usually make use of memory, and the data is growing exponentially. Through situations where there are enormous amount of data, the applications will have to decrease the size of the evaluation windows, and it will be able to affect negatively at the final application result;
- **Storage**: Data storage is one of the most important challenges of Big Data. It is the point that has been getting the attraction of researchers in recent years, especially in the batch-processing area [5, 2]. It involves the data placement and movement, especially problems with security. Although, both have impacts on the latency of execution;
- **CPU**: Clouds usually are environments that share the computational resource and also have limitations between the users (e.g., 300 cores in Microsoft Azure) [13]. These conditions will influence in the creation of the flows on the Stream-processing;

The structuring of a geographically distributed environment will assist in decreasing the restrictions given in a single site. Although, problems will arise related to data placement and data movement. The main one is the use of the Internet for the communication. In this scenario, it is not possible to define the time for the data transfer since the bandwidth and latency have wide fluctuations.

Therefore, methods and techniques should be evaluated to address the barriers of infrastructure. In the Stream-processing environments, there is a strong constraint related to the processing time. The result should be obtained in seconds or milliseconds (i.e., real-time or near real-time).

## 2.2. Stream-Processing Models

An important point that will influence the running time is the way the application will handle the data. Furthermore, as presented studies ([3] and [14]), an opportunity to overcome it is through the accumulation of the events and transfer them in data blocks (i.e., considering the time constraints given by the application). So, the latency cost of the transfer will be lower and the throughput of the events by time will be higher.

So within the options of Stream-Processing models, there are the following strategies of event aggregations: **One-at-a-time**, most of it is used to the real-time constraint. The events are transmitted directly to the Message Broker as are generated; **Batch-sized** used to the near real-time constraint. Events are retained in the source and sent to the Broker in batches.

Based on the mentioned techniques, it is important to select one that may improve the performance in proposed environment. So, we evaluate those approaches in a cloud infrastructure.

The accumulation of events will be viable in the process since it is intended to work with near real-time environments. By using batched-size model is expected to identify the best amount to accumulate or even check the interference the Internet in the environment. For the One-at-a-time approach each event will be added the network latency since it will be added only once in a Batch-sized approach.

In this study, the "mini-batch" size will vary primarily to verify the accumulation time and the impact caused by the application. To assess these impacts will use a fully distributed environment as shown in Figure 1 and seek to be identify the interference in the application run-time.

# 3. Decoupled Scenarios Performance Analysis

This section describes the evaluation and methodology to demonstrate the impacts of Stream-Processing Model on Multi Geo-spatial Environment. Moreover, we present the utilized tools, collected metrics and a performance analysis.

## 3.1. Methodology

We aims to measure all points within the decoupled scenario to be able to identify possibles bottleneck. The measured metrics were the completion time and the total number of events to verify each approach and, also we assessed the following points: events per seconds; Flink's events throughput; duration of each event;

The experiments comprise empirical evaluation performed on the Microsoft Azure cloud, at the PaaS level, using a VM on West of US as the Data Source, a VM on North of EU as the Message Broker and three VMs on East of Japan as the Stream-processing set. The system runs on A10 instances (processor Intel Xeon E5-2670 @ 2.6 GHz, 8 cores, 56GB DDR3-1600 MHz RAM and 120 GB storage) with Ubuntu Server. Each VM instance synchronizes

its time with the NTP Server. Experiments were designed based on the methodology proposed by [8] with a replication factor equal to 20 and 7 batch sizes. The experiments were performed as the following tools: **Data Source**: A Java implementation to produce parallel events; **Message Broker**: Apache Kafka 0.10.0; **Stream-processing Engine**: Apache Flink.

The stream-processing models evaluated were **One-at-a-time**: one event at a time and the **Batch-based**: batches with 2, 10, 150, 250, 500 and 1000 events. In order to evaluate the proposed environments the following tools were used: Iperf to measure the throughput and the bandwidth among the evaluated regions; Dstat-monitor was used for measuring resource consumption, CPU utilization, memory usage, disk and network I/O; Hping was used to obtain information regarding the network latency between the test environments; Ganglia has been used to monitor the reception and sending messages time through Apache Kafka and Flink.

For our experiments, we used a sentimental analysis application, as our Stream Processing Motor, where tweets were classified. The application derivates from a Natural Processing Language algorithm, based on a dictionary. Tests were performed in 140 runs, where the size of the tweet was $\approx 2KB$. In each run were sent 1.000,000 tweets where the experiment time took $\approx 12\ hours$.

### 3.2. Results

Figure 3 shows the entire execution over the decoupled environment, over the complete execution over all proposed scenarios, i.e., batch sizes. We noticed a significant variation in sending smaller batches. Most of it, caused by the substantial number of messages sent over the network, being more susceptible to failures and high latency that is caused mainly by the cloud services or the network links. Although, bigger batches performed less variation and

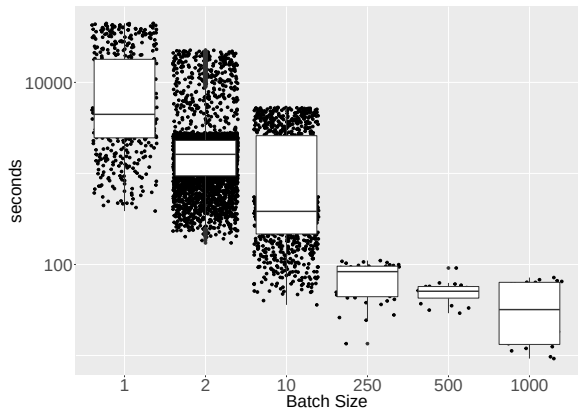



**Figure 3. Execution time over the entire model**

lower execution time. We can observe that in bigger batches there is fewer event consumption but with smaller processing time. Showing that for that type of environment is viable to send bigger messages than send more amounts of smaller messages. Batch sizes of 250 to 1000 shown promising results, providing stability and a shorter run time.

Since we adopt the use of a synthetic uniform data set, it has expected that the execution time may increase proportionally. However, it does not occur. The warm-up and cool-down execution of Apache Flink explain that behavior, which is the time of deployment of a new job and refresh of the framework.

Obviously, we also need to consider as well the characteristics of the evaluated application, that may benefit the approach. For this reason, the chosen application is I/O bound. It is because most of the Big Data Stream applications are considered I/O bound [7] the strategy may benefit most of the applications.

Moreover, it is possible to conclude that the aggregation is feasible since there is no further increase in execution time inside Flink. It also may prove, that there is an improvement of exploitation from the available resources since the throughput increased, but it needs further and deeply evaluations.

As seen in Figure 3 the batch sizes of 1 to 10 are significantly inferior in execution time, since the involved logistics to send plenty of messages besides using a bigger batch with multiples streams. At this point, Figure 4 reinforces that idea, considering that it presents the entire throughput time to prepare, send and process a batch-sized.

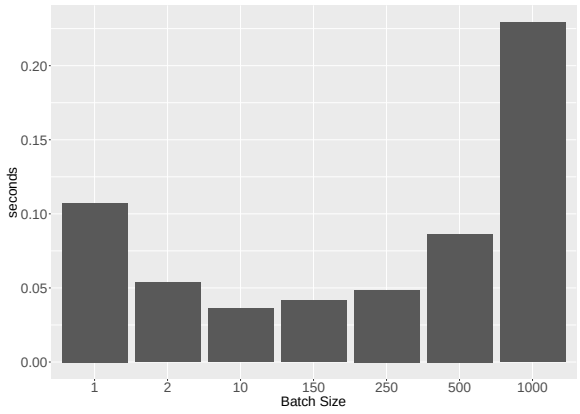



**Figure 4. Event time**

The Figure 4 shows the mean time of an event per each batch size. From the batch of size one to ten there's a time decrease of 66% that is a quite promising result. From the batch size of 10 to 500 it is possible to notice an increase in time. From this point we can see that the strategy is no longer efficient. Whatsoever, this is not a considerable growth since the amount messages almost doubles, comparing directly with the batch size of one message at a time. Conclusively, it is also possible to notice the degradation caused by the strategy of 1000 events in a batch. It is caused mostly by the overhead to aggregate that amount of events in memory.

## 4. Related Work

JetStream [14] is a set of strategies for efficient transfers of events between cloud data-centers. JetStream can self-adapt to the streams conditions by modeling and to monitor a set of context parameters. It further aggregates the available bandwidth by enabling multi-route streaming across the cloud sites. The research focus on events transfers between inter and intra-nodes. They propose an adaptive Cloud batching; the algorithm aggregates the streams

in batches in latency reduction. The main idea of JetStream is interesting. However, it just considers the latency and not the volatility. The proposal target environment where computational resources can leave unexpectedly and the scheduling policies must adapt to it.

In [9] are introduced solutions to enable elasticity over shared clusters. Mainly focused on the demand where the input rate of streams may vary drastically from each location. A monitor is proposed that discover bottlenecks in the stream data flow. Rescales are done in the capacity of processing seeking to solve the found bottlenecks. It concentrates principally on the capacity of processing, not the data itself, as proposed in this work.

SMART [4] is a framework to offer an efficient development of Big Data analysis for small and medium-sized organizations. SMART consider heterogeneous data sources and the data analysis focus on Geo-Distributed data, consider cost, failure probability, network overhead, I/O throughput and minimize the transfers between the computational resources. These parameters are not enough to work with Stream-processing and heterogeneity. To overcome the environment limitations, it is necessary to input information from the devices, such as memory, CPU speed, and storage.

Similarly, [11] purpose a generic, extensible, scalable, fine-grained re-configurable deployment and multi-cloud framework. It is based on a lightweight kernel and provides a hierarchical DSL (Domain Specific Language). DSL allows it to achieve a fine-grained level of administration. Counterpart the solution does not control the workload at the nodes and the possible failures, the Deployment Manager is just an interface to set the devices and the Virtual Machines.

## 5. Conclusions

Our findings demonstrate that it is necessary to evaluate the environment and the techniques used for sending batches, since it may vary according to network latency and environmental characteristics. Results proved that with small batches there is a wide variation in the time of the messages and possibly a high time within the duration of the event. In contrast to larger batches the number of messages is smaller, with less variability and preventing possible failures, thus may have a slight variation in time of the messages with relatively smaller durations.

Therefore, methods and techniques should be evaluated to address the barriers of infrastructure. In the Stream-processing environments, there is a strong constraint related to the processing time. The result should be in seconds or milliseconds (i.e., real-time or near real-time). As future work, a proposal of self-adaptative strategy may be made, to dynamic define size to the batching approach based on the constraints seen in those environments and following the necessity of each application.

## References


[1] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinländer, M. J. Sax, S. Schelter, M. Höger, K. Tzoumas, and D. Warneke. The stratosphere platform for big data analytics. *The VLDB Journal*, 23(6):939–964, Dec. 2014.

[2] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. B. Zdonik. Scalable distributed stream processing. In *CIDR*, volume 3, pages 257–268, 2003.

[3] T. Das, Y. Zhong, I. Stoica, and S. Shenker. Adaptive stream processing using dynamic batch sizing. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 16:1–16:13, New York, NY, USA, 2014. ACM.

[4] J. C. S. dos Anjos, M. D. Assuno, J. Bez, A. Carissimi, J. P. C. L. Costa, F. Freitag, V. Markl, P. Fergus, R. Pereira, E. P. de Freitas, G. Fedak, and C. F. R. Geyer. Smart: An application framework for real time big data analysis on heterogeneous cloud environments. In *In proceedings of 15th IEEE International Conference on Computer and Information Technology (CIT-2015), Liverpool, England, UK, October 2015*. IEEE Computer Society, 2015.

[5] M. Garofalakis, J. Gehrke, and R. Rastogi. *Data Stream Management: Processing High-Speed Data Streams*. Springer, 2016.

[6] L. Gu, D. Zeng, S. Guo, Y. Xiang, and J. Hu. A general communication cost optimization framework for big data stream processing in geo-distributed data centers. *IEEE Transactions on Computers*, 65(1):19–29, Jan 2016.

[7] B. Huang, S. Huang, J. Dai, J. Huang, and T. Xie. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*, 00:41–51, 2010.

[8] R. Jain. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing. Wiley, 1991.

[9] J. Li, C. Pu, Y. Chen, D. Gmach, and D. Milojicic. Enabling elastic stream processing in shared clusters. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 108–115, June 2016.

[10] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 170–177, Dec 2010.

[11] L. M. Pham, A. Tchana, D. Donsez, V. Zurczak, P. Y. Gibello, and N. de Palma. An adaptable framework to deploy complex applications onto multi-cloud platforms. In *Computing Communication Technologies - Research, Innovation, and Vision for the Future (RIVF), 2015 IEEE RIVF International Conference on*, pages 169–174, Jan 2015.

[12] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy. Storm@twitter. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 147–156, New York, NY, USA, 2014. ACM.

[13] R. Tudoran, A. Costan, and G. Antoniu. Overflow: Multi-site aware big data management for scientific workflows on clouds. *IEEE Transactions on Cloud Computing*, 4(1):76–89, Jan 2016.

[14] R. Tudoran, O. Nano, I. Santos, A. Costan, H. Soncu, L. Bougé, and G. Antoniu. Jetstream: Enabling high performance event streaming across cloud data-centers. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS '14, pages 23–34, New York, NY, USA, 2014. ACM.

[15] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[16] Z. Zhuang, T. Feng, Y. Pan, H. Ramachandra, and B. Sridharan. Effective multi-stream joining in apache samza framework. In *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 267–274, June 2016.

# BIGHybrid: a toolkit for MapReduce in hybrid environments

Vinicius P. Perego, Kassiano J. Matteussi, Julio C.S. Anjos, Claudio Geyer
Institute of Informatics, UFRGS
Bento Gonąlves Avenue, 9500, Porto Alegre, RS, Brazil
vpperego@inf.ufrgs.br

## Abstract

*The high availability of Cloud Computing attracts users to deploy and run Big Data applications. Unfortunately, its usage implies in a huge price for the resources allocated to the job. Contrarily, heterogeneous environments like Desktop Grids also facilitates the processing of high datasets due to its large number of free resources. In order to reduce the costs with Cloud Computing, this work proposes a solution based on a hybrid architecture, which distributes data in the Cloud and Volunteer Computing. Alongside the hybrid model, a toolkit called BIGHybrid was developed to evaluate solutions in a hybrid environment.*

## 1. Introduction

The increase in data consumption by modern applications requires that distributed systems have resources available to process it. In this scenario, Cloud Computing has become the most sought solution due to its availability. However, cloud service providers charge the users by resources used, which difficults the use of this type of infrastructure for processing high amounts of data. Alternately, Desktop Grid (DG) can be a good solution to execute these applications with additional power donated by idle computers (volunteer computing). The problem with this source of processing power is the volatility that needs an improved fault tolerance mechanism. To increase the resource pool, one could use a hybrid solution based on Cloud and Desktop Grid.

Hadoop [9] is a framework that implements MapReduce [5] , a programming model which eases the creation of distributed programs by abstracting the parallelization and achieving some degree of fault tolerance with its built-in algorithms.

This work proposes an update to the original Hadoop aiming to use it as a hybrid infrastructure based on Cloud Computing and DG. The changes in the Hadoop algorithms were made in a toolkit written over SimGrid.

SimGrid [4] is a toolkit for the simulation of distributed systems, enabling the user to define the execution platform, scheduling algorithms and take measures like the execution time.

## 2. Related Works

GroudSim [7] is a simulator toolkit for Grid and Cloud focusing on scientific applications. It is based on scalable discrete-event and supports traces from both application, host and event traces. Unlike BIGhybrid, GroundSim does not support hybrid or volatile simulations.

CloudSim[3], an extension of GridSim, focuses on Cloud simulation. It allows cloud service modeling such as service brokers, provisioning and allocation policies at a single physical node. However, CloudSim can not generate traces and simulate data-intensive applications, which are main features of BIGHybrid.

The AweSim simulator [8] is based on a network simulation framework that involves a fine-grained simulation for workflow computation and data movement across multiple Clouds. The implementation uses workload traces from a production data analysis service and is thus similar to the BIGhybrid simulator that adopts its behavior from traces in a volatile real-world environment.

## 3. Objectives

This work proposes a toolkit to simulate MapReduce in hybrid environments, adapting the original Hadoop MapReduce with enhances on its algorithms and additional inclusions to work with volunteer computing. BIGHybrid enables the evaluation of the impact of the changes on the job configuration or resources (e.g., network bandwidth, latency and processors flop power).

## 4. Model

BigHybrid [1] is organized in a layered architecture pattern as shown in Figure 1 . The System Config module al-

lows the user to specify the workload (e.g., chunks size and data distribution among cloud and DG) and the platform for Simgrid of both cloud and DG environments.

In the lower layer, the middlewares BitDew-MapReduce and Cloud-BlobSeer control the DG and Hadoop resources, respectively. BlobSeer is a DFS that manages a huge amount of data in a flat sequence of bytes called BLOBs (Binary Large Objects). BlobSeer extends the Hadoop file system (HDFS) by enabling concurrent access to metadata and high throughput with concurrent reading, writing, and updating of the data. This incremental update is necessary for data management in a hybrid infrastructure.

Bit-Dew exploits protocols like P2P, HTTP, BitTorrent and FTP to data management in Grids. The BitDew-MapReduce, a MapReduce implementation adapted to a volatile environment, contains important features like fault-tolerance mechanism, data placement and incremental update which are important for hybrid infrastructure.

## 5. Prototype

BIGhybrid was implemented over the Simgrid toolkit. It relies on MRA++ [2] for BitDew-MapReduce and MRSG [6] for Cloud-BlobSeer. The original version of MRA++ was extended for the purpose of simulating Volunteer Computing, where the volatility module was added. All the simulators were developed with Simgrid through the MSG application programming interface.

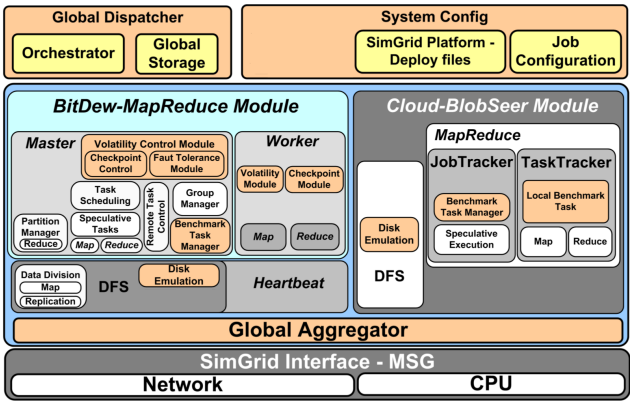



**Figure 1. BIGHybrid architecture**

## 6. Results

The hardware configuration is based on the computational capacity of processors equivalent to an Intel Xeon - 2 Cores, 4M Cache, 2.13 GHz 5E+09 Flops and the heterogeneous environment with its capacity distributed between 4E+09 to 6E+09 Flop. The networking bandwidth for Cloud is 1 Gbps and to DG is 100 Mbps.

An experiment was made in a low-scale hybrid environment with the results in Figure 2. The line red indicates time execution of 200 tasks (chunk size 64 MB) on 128 machines in a single Cloud deploying, equivalent to 503 seconds. Each execution distributes the machines and the workload in different scenarios. Executions A, B and C show a gain for the hybrid execution over the original cloud deployment.

## 7. Conclusion

In this study, the characteristics of a hybrid infrastructure were investigated, addressing problems to adapt the MapReduce to execute in these scenarios. It was possible to achieve an acceptable execution and even gain some performance with the low-scale scenario tested. Some features like I/O contention need to be added to increase the accuracy of the Cloud simulation. Although it is possible to reduce the overall resource allocation in the Cloud and consequently the cost, it is still necessary to measure the gains with the hybrid solution.

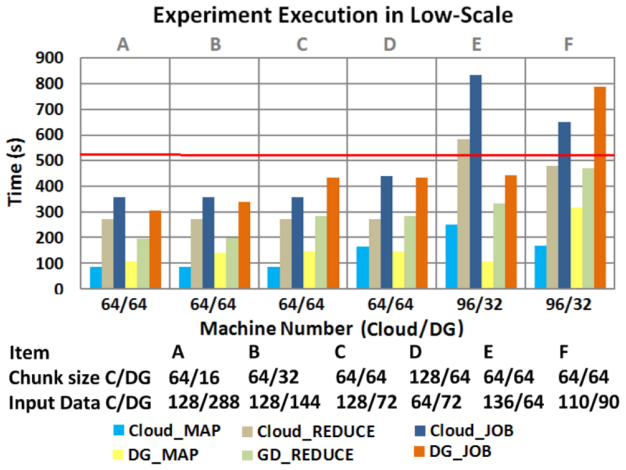



**Figure 2. Low-Scale hybrid environment**

## References


[1] J. Anjos, G. Fedak, and C. F. Geyer. Bighybrid: a simulator for mapreduce applications in hybrid distributed infrastructures validated with the grid5000 experimental platform. *Concurrency and Computation: Practice and Experience*, 28(8):2416–2439, 2016.

[2] J. C. Anjos, I. Carrera, W. Kolberg, A. L. Tibola, L. B. Arantes, and C. R. Geyer. Mra++: Scheduling and data placement on mapreduce for heterogeneous environments. *Future Generation Computer Systems*, 42:22–35, 2015.

[3] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[4] H. Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *Cluster computing and the grid, 2001. proceedings. first ieee/acm international symposium on*, pages 430–437. IEEE, 2001.

[5] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[6] W. Kolberg, P. D. B. Marcos, J. C. Anjos, A. K. Miyazaki, C. R. Geyer, and L. B. Arantes. Mrsg–a mapreduce simulator over simgrid. *Parallel Computing*, 39(4):233–244, 2013.

[7] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer. Groudsim: an event-based simulation framework for computational grids and clouds. In *European Conference on Parallel Processing*, pages 305–313. Springer, 2010.

[8] W. Tang, J. Jenkins, F. Meyer, R. Ross, R. Kettimuthu, L. Winkler, X. Yang, T. Lehman, and N. Desai. Data-aware resource scheduling for multicloud workflows: A fine-grained simulation approach. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 887–892. IEEE, 2014.

[9] T. White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

# A Pre-Aggregation Strategy Towards Reduction on Job Completion Time in Stream Process Systems

Breno Fanchiotti Zanchetta
Federal University of Rio Grande do Sul
Institute of Informatics
Room 205 – ZIP 15.064 – 91.501-970
Porto Alegre – RS – Brazil
bfzanchetta@hotmail.com
bfzanchetta@gmail.com

Paulo R. Souza Jr, Kassiano J. Matteussi
Vincius P. Perego, Julio C. S. Anjos
Claudio F. R. Geyer, Edison P. de Freitas
Federal University of Rio Grande do Sul
Institute of Informatics
Room 205 – ZIP 15.064 – 91.501-970
Porto Alegre – RS – Brazil
{prrsjunior, kjmatteussi, vpperego}@inf.ufrgs.br
{jcsanjos, geyer, edison.pignaton}@inf.ufrgs.br

## Abstract

*A very essential part of science of Big Data is called streaming, and treats a very small-sized data called Stream, and possesses real real time processing requirements. There are several economic real world applications that make use of this tool in order to retrieve high trending topics and make economic decisions. Regarding the most known techniques, this aggregation technique can be labeled as batching and implies in real gains on various applications fidelity with small latency trade-offs. This work aims to test many solutions of pre-aggregation in an initial controlled environment and further test it on a network-stressed scenario, towards the mission of finding the best topology configuration that reduces network congestion. Every test was performed in a private cluster and the results point towards a set of values that generate real time gains on Flink's processing motor for given intervals of configuration variables.*

*Index Terms: Big Data, Stream Processing, Data Aggregation.*

## 1. Introduction

Big Data is a term in evidence in communication media such as Facebook and Twitter, which make use of this topic in their daily activities. Current statistics found in [7] presents data numbers estimatives: Facebook gets 4.75 billion shares, 4.5 billion likes, 420 million status updates and 300 million photos every day. This scenario generates tremendous amounts of data that have different types, sizes and importances. Current Big Data solutions are divided mainly into Stream processing and Batch processing. The first is used for smaller pieces of information with high income rates and, the second uses bigger sized data with slight smaller input flow.

The basic unit in Stream Processing is called stream, and may contain different purposes depending on the system that produced it. For instance, [6] focuses a new approach and language on airplane error calculation using streams generated by the airplane error logs. This approach used Streams in order to send error logs from airplane coupled sensors to its main computer for calculations that involve planes angle correction, altitude, temperature and others.

Even though it demands less complexity than a real time airplane software that prevents error calculations, Twitter still is very important in order to retrieve trending topics and other economic tendencies, as mentioned by [1]. Also, tweets consist of easily accessible data with very small size, therefore it is used in this work as dataset.

There are two main strategies on streaming: sending one tuple at a time or aggregating multiple streams into batches and sending them for processing. Roughly, aggregating streams technique show execution gain because this strategy reduces context switch on the big data processing engine. Also, it increases throughput and latency a priori with increases on the applications fidelity.

Its important to explain the feature called Window, which is an engine feature that allows its running application to perform the operators calculations together in time intervals like a physical sliding window, reducing the overhead on the application. Allied to the window, this work proposes a prototype that tests if there is an optimal aggregation variable size that generates Flinks time reduction, also aiming reduction of bandwidth latency and achieving an increase in throughput.

This work is structured in background and related work, followed by third section which specifies the methods used in this work and the proposed solution. Section four demonstrates results, followed by section five with discussions of these results. Finally, there is section six with future work trends.

## 2. Background and Related Work

### 2.1. State-of-the-art

The work contained on [2] serves as a guideline for stream application development. It approaches 8 basic rules that every stream processing environment must attend in order to respect real time requisites. This work brings more attention towards the fourth rule: The requirement of real time stream processing is that a developed engine must guarantee predictable and repeatable outcomes.

In order to attend this importance, a comparison metric was chosen to compare final output logs from each and every execution to maintain a certain level of repeatable outcomes. Logically, in real world applications there is no true way to predict stream input order, however, as [6] states, during tests a few guidelines must be respected in order to maintain this real time feature.

Also, as [3] points out, there is no standardized course of action for designing a topology for streaming applications, but a priori. One of the works contribution is justified by this argument, since it provided experiments based on empirical choices that were reassured by [8], [4] and [5]. These works also developed stream aggregation topologies without specifying the strategy behind them. All these facts reassure this paper work intents to run tests based on empirical evaluation.

The work found at [2] reduces computational cost by using a single persistent random variable that marks the lifetime of each key on the cache memory. This work does not intend to include either cached memory or time variables in stake, since time management demands clock synchronization and making usage of cached memory is not trivial. Meanwhile, the cluster random access memory is a brand resource, it must be dealt accordingly.

The work described in [10] shows how event-based stream processing may differ from aggregation technique, expliciting that the second technique is able to provide gain in executions whenever there are events out-of-order from more than one source. This was used as basis for this work, mainly because singled source datasets proved to be less afected by aggregation strategy in most of the test case scenarios and thus, the validation of this prototype might be accessed by the use of different dataset delivery strategies.

### 2.2. Related Work

The work contained in [9] proposes a streaming platform in real time. It tries to determine the optimal batching values in order to increase network transfer rates between multiple routes. It makes good usage of the fact that some big data providers dont charge fees to provide intranodal data transfers but charge internodal transfers.

In order to validate their hopes of reducing transfer times, their prototype perform intranodal replications and recalculate latency between the new nodes and distributed applications. Its possible that some of the new replications contain smaller latencies than the previous original data. Therefore, tests that oscillate batching sizes are performed for each of these new replications in order to detect the quickest path and best batching size.

All these calculations are performed autonomously and the system self-adapts to changes in the network, choosing the new best batching size and best route based on latency exhausting calculations. Our prototype differs from this work because it seeks to find optimal values overall for a given configuration in order to perform future tests for generic environments. Also, the previously mentioned work performs a series of calculations, batching reordering, replications and other techniques that dont convince their gains.

## 3. Materials

### 3.1. Methods and Model

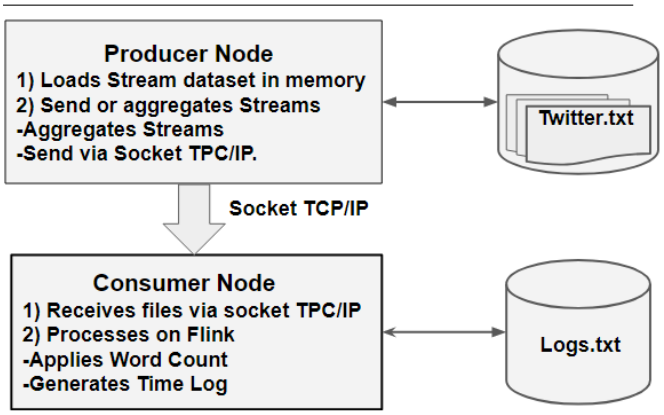



**Figure 1. Model**

This works used real world dataset, pre-collected by Twitters API in java, containing multiple tweets. Also, the dataset received a few documents from USA criminalistic department and AIDS statistics from health department.

### 3.2. Implementation

The TCP/IP protocol was used in the Python code in order to stablish communication with the processing engine and feed the dataset to it. First node of the private cluster performs the data aggregation and send it for validation job on second node. This work chose Word Count application in order to answer a question: Is there a value for pre-aggregation that generates execution time gain on [1]Flink?

In order to perform experiments on streaming, this work chose Apache Flink software to be used as stream processing engine because of it's easy to be configured and used.The earlier implementation was performed in Java language. Since procedures that involve high I/O operations with stream aggregation resulted in poor outcomes, this work moved towards a Python approach and started to show promising results.

The pre-aggregation class was developed in Python, configured to receive aggregation sizes per stream that comes in. As a single node of the cluster was configured to be the stream producer and the second was configured to hold the Flink as the consumer node.

It was set a fixed number of lines as a aggregation parameter to prove that pre-aggregation influences Flink-window processing time positively. For every stream that the Python nodes receives, it aggregates a fixed number of lines until the stream is all read. If the stream is over, the pre-aggregation fetches other streams in order to fill the entire aggregation variable.

**Result:** Aggregation time on Python,
Stream processing time on Flink
Initializes n aggregation size;
Initializes concatenation string;
Starts time initialization;
**while** *has stream file* **do**
  **while** *file has next line* **do**
    Concatenates line to String;
    **if** *String greater than n* **then**
      Send String to Flink ;
      Restarts String ;
  **end**
**end**

**Algorithm 1:** Aggregation Strategy

Finally, when this aggregation variable fills entirely, the concatenation is sent. Concerning configurations, 33 repetitions of each test were set for each possibility of variable permutation, including different aggregation sizes, 5 second window or not, with different sizes of parallelism, on the two given datasets. The experiment has 6 strategies of aggregation, 2 datasets, 2 types of window and 4 sizes of parallelism. Permuting these factors, the entire experiment had 6x2x2x4=96 possibilities, which the 33 experiments ran singularly, in order to achieve a close behaviour of the desired pattern (so 96x33=3168 test case total).

1 https://flink.apache.org/

### 3.3. Equipment

The tests were executed on a private Cluster, with 5x Power Edge 1950 8 Cores (4 Real and 4 Virtual) containing 16GB of RAM. LAN Network bandwidth is known to be 100 Mbps and it was used Ubuntu Server version 16.04 LTS.

## 4. Results

Results showed that small consistent datasets with no Flink window had the execution time varied negatively for all values in the training group, with one exception of the largest aggregation size (1000 concatenation size). The larger 1,6GB dataset showed that there is a slight advantage when performing aggregation with 500 units of concatenation in average.

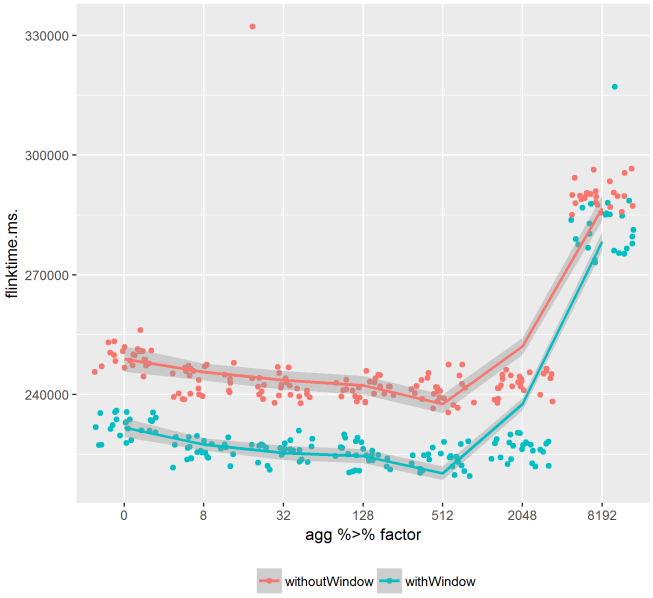



**Figure 2. Big Dataset Jitter**

On the other hand, Flink 5-seconded window experiments showed different results. For bigger 1,6GB dataset any aggregation showed a slight gain in execution up to 750ms. About the last test, using a 121,7MB with the same 5 second Flink window, there was an increase in processing time.

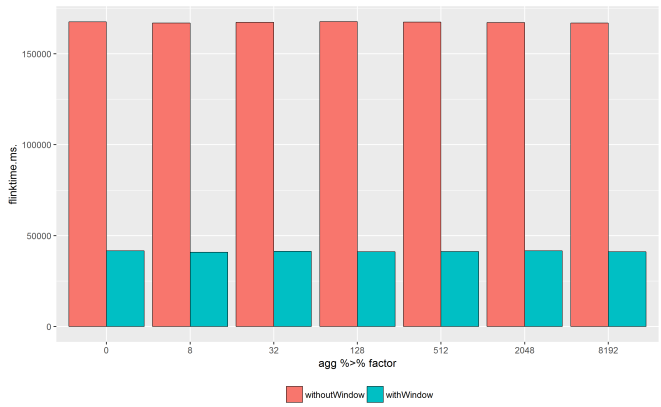



**Figure 3. Tiny Dataset Jitter**

Figure 1 points that for bigger datasets there is an optimal value of 512. On the other hand there is a strong effort to avoid high values of batching and no aggregation, since these strategies obtained worse results. This was visible for any value higher than 512. Figure 2 points that for smaller datasets there is also a high tendency to increase time in the engine when using a window regardless of the agreggation strategy. However, for multiple aggregation sizes on very small datasets the results were very insignificant and didn't pose to show any relative gains.

## 5. Discussion

At the beginning, this work tried to detect a relation between stream pre-aggregation and time gain on Flink word count application given two real world datasets. In order to test this work model, multiple runs were executed on a couple of nodes of a private cluster for the given dataset and distinctive variables of configuration. It was shown that for greatter datasets, the engine benefited from aggregation strategies and window, with best value of 512 for the given problem in the given topology. For smaller datasets, there was a gain in window use, but no significant gain in aggregation strategy.

## 6. Conclusion and Future Work

Future work will focus on developing dynamic strategies of updating variables settings which might be accomplished by dynamic tables. Whilst this approach focuses on tabled methods, there is also room for insertion of probabilistic models or machine learning.

Also, some changes might be done towards measurement of throughput and latency as secondary metrics. The implementation of a simulator might accomplish more results in this case.

## References

[1] P. Basanta-Val, N. Fernandez-Garcia, L. Sanchez-Fernandez, and J. A. Fisteus. Patterns for distributed real-time stream processing. *IEEE Transactions on Parallel and Distributed Systems*, 2017.

[2] N. Duffield, Y. Xu, L. Xia, N. Ahmed, and M. Yu. Stream aggregation through order sampling. *arXiv preprint arXiv:1703.02693*, 2017.

[3] R. Guerraoui, E. Le Merrer, R. Patra, and B.-D. Tran. Frugal topology construction for stream aggregation in the cloud. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. Ieee, 2016.

[4] Q. Huang and P. P. Lee. Ld-sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams. In *INFOCOM, 2014 Proceedings IEEE*, pages 1420–1428. IEEE, 2014.

[5] R. Huebsch, M. Garofalakis, J. M. Hellerstein, and I. Stoica. Sharing aggregate computation for distributed queries. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 485–496. ACM, 2007.

[6] S. Imai, E. Blasch, A. Galli, W. Zhu, F. Lee, and C. A. Varela. Airplane flight safety using error-tolerant data stream processing. *IEEE Aerospace and Electronic Systems Magazine*, 32(4):4–17, 2017.

[7] D. Noyes. The top 20 valuable facebook statistics–updated february 2015. *Retrieved from Zephoria: https://zephoria.com/social-media/top-15-valuable-facebookstatistics*, 2015.

[8] O. Papapetrou, M. Garofalakis, and A. Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *Proceedings of the VLDB Endowment*, 5(10):992–1003, 2012.

[9] R. Tudoran, A. Costan, O. Nano, I. Santos, H. Soncu, and G. Antoniu. Jetstream: Enabling high throughput live event streaming on multi-site clouds. *Future Generation Computer Systems*, 54:274–291, 2016.

[10] K. Wahner. Wahner, kai. "real-time stream processing as game changer in a big data world with hadoop and data warehouse." internet. 2014. [Online]. Acessado em: 30-06-2017.

# Using Simulation to Evaluate and Tune the Performance of Dynamic Load Balancing of an Over-decomposed Geophysics Application

Rafael Keller Tesser
Informatics Institute UFRGS
rktesser@inf.ufrgs.br

Lucas Mello Schnorr
Informatics Institute UFRGS
schnorr@inf.ufrgs.br

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

Fabrice Dupros
BRGM, Orlans, France
f.dupros@brgm.fr

Philippe O. A. Navaux
Informatics Institute UFRGS

## Abstract

*Finite difference methods are commonplace in scientific computing. Despite their apparent regularity, they often exhibit load imbalance that damages their efficiency. We characterize the spatial and temporal load imbalance of Ondes3D, a seismic wave propagation simulator. We reveal that this imbalance originates from the nature of the input data and from low-level CPU optimizations. Such dynamic imbalance should therefore be quite common and is intractable by any static approach or classical code reorganization. An effective solution, with few code modifications, combines domain over-decomposition and dynamic load balancing (e.g., with AMPI), migrating data and computation at the granularity of an MPI rank. It generally requires a careful tuning of the over-decomposition level, the load balancing heuristic and frequency. These choices are quite dependent on application and platform characteristics. In this paper, we propose a methodology that leverages the capabilities of the SimGrid framework to conduct such study at low experimental cost. It combines emulation, simulation, and application modeling that requires minimal code modification and yet manages to capture both spatial and temporal load imbalance, faithfully predicting its overall performance. We compare simulation and real executions results and show how our strategy can be used to determine the best load balancing configuration for a given application/hardware configuration.*

# Analyzing Dynamic Task-Based Applications on Hybrid Platforms: An Agile Scripting Approach

Vinicius Garcia Pinto
Informatics Institute
Federal University of Rio Grande do Sul

Luka Stanisic
Inria Bordeaux Sud-Ouest
France

Arnaud Legrand CNRS
Univ. Grenoble Alpes
France

Lucas M. Schnorr
Informatics Institute
Federal University of Rio Grande do Sul

Samuel Thibault
Inria Bordeaux Sud-Ouest
France

Vincent Danjean
CNRS Univ. Grenoble Alpes
France

## Abstract

*In this paper, we present visual analysis techniques to evaluate the performance of HPC task-based applications on hybrid architectures. Our approach is based on composing modern data analysis tools (pjdump, R, ggplot2, plotly), enabling an agile and flexible scripting framework with minor development cost. We validate our proposal by analyzing traces from the full- fledged implementation of the Cholesky decomposition available in the MORSE library running on a hybrid (CPU/GPU) platform. The analysis compares two different workloads and three different task schedulers from the StarPU runtime system. Our analysis based on composite views allows to identify allocation mistakes, priority problems in scheduling decisions, GPU tasks anomalies causing bad performance, and critical path issues.*

# Performance Prediction of Stencil Applications based on Machine Learning

Víctor Martínez and Philippe Navaux
Informatics Institute (INF),
Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre, Brazil
{victor.martinez, navaux}@inf.ufrgs.br

## Abstract

*Stencil computations are the basis to solve many problems related to Partial Differential Equations (PDEs). Performance prediction for such numerical kernels is a major issue as many critical parameters (architectural features, compiler flags, memory policies, multithreading strategies) are involved. In this context, fast and accurate seismic processing workflow is a critical example for this kind of computations. In order to understand complex geological structures, the numerical kernels used mainly arise from the discretization of Partial Differential Equations (PDEs) and High Performance Computing methods play a major in seismic modeling. This paper focuses on the use of Machine Learning to predict the performance of stencil kernels on multi-core and many-core architectures. Low-level hardware counters (e.g. cache-misses and TLB misses) on a limited number of executions are used to build our predictive model. We have considered three different kernels (7-point Jacobi, acoustic and seismic wave modeling) to demonstrate the effectiveness of our approach. Our results show that performance can be predicted by simulations of hardware counters with high accuracy.*

## 1. Introduction

Stencil computations lie at the heart of many problems in areas as diverse as electromagnetics, fluid dynamics or geophysics. The trend for High Performance Computing (HPC) applications is to pay a higher cost in order to optimize the overall performance. This comes from the complexity of many interdependent factors (non-uniform memory access, vectorization, compiler optimizations, memory policies) at an architectural level that may severely influence the application's behavior.

On the one hand, wave propagation modeling is the current backbone for several seismic tools. It has been extensively applied for imaging potential oil and gas reservoirs beneath salt domes for the last five years. Such acoustic propagation engines should be continuously ported to the newest HPC hardware available to maintain competitiveness. At the same time, on the HPC hardware front, the days of faster single core CPUs are over, and the solutions adopted are being replaced by many-core technologies [5, 4]. In this context, several heuristics or frameworks have been proposed to speed up the process of finding the best configuration for stencil applications [3, 14, 17, 12].

On the other hand, Machine Learning (ML) is a comprehensive methodology for optimization that could be applied to find patterns on a large set of input parameters. Recently, ML algorithms have been used on HPC systems under different situations. In [19] the authors used ML algorithms to select the best job scheduling algorithm on heterogeneous platforms whereas in [2] the authors proposed an ML-based scheme to select the best I/O scheduling algorithm for different applications and input parameters. And recently, in [13] the authors used ML algorithms to predict the performance of stencil computations on multicore architectures.

In this paper, we describe the general procedure to build a suitable ML-based performance model for classical numerical kernels: 7-point Jacobi and acoustic and seismic wave modeling. This model allows us to simulate the performance behavior of stencil computations on multi-core architectures. The paper is organized as follows. Section 2 provides the fundamentals of stencils under study. Section 3 describes the methodology of our ML-based approach. Section 4 presents experiment configuration and model accuracy. Section 5 describes related works, and finally Section 6 concludes this paper.

## 2. Stencil models

From the numerical analysis point of view, stencil-based computations often arise when discretizing Partial Differential Equations (PDEs). For instance, the Finite-Difference Methods (FDMs) computational procedure consists in us-

ing the neighboring points in the north-south, east-west and forward-backward directions to evaluate the current grid point in the case of a three-dimensional Cartesian grid. The algorithm then moves to the next point applying the same stencil computation until the entire spatial grid has been traversed. In this work, we study two well-known stencil kernels:

1. **7-point Jacobi**: The seven-point Jacobi stencil is a reference example of numerical kernel used in various context in order to evaluate the impact of advanced reformulation or the impact of the underlying architecture. A review can be found in [7] for instance.
2. **Seismic Wave**: Evaluation of damages occurred during strong ground motion is critical for urban planning. The numerical kernel under study relies on the classical 4-th order in space and second-order in time approximation and was detailed in [18, 15, 10].
3. **Acoustic wave**: We consider the isotropic acoustic wave propagation under Dirichlet boundary conditions over a finite 3D rectangular domain. The operator is discretized by a $12^{th}$ order finite differences approximation and the time derivatives are approximated by a $2^{nd}$ order finite differences operator. This kernel represents the cornerstone of the classical Reverse Time Migration imaging procedure. For sake of clarity of this paper focused on the impact of machine learning methodology, we will not go into too many details regarding the implementation and interesting readers can refer to [1].

## 3. Machine Learning Methodology

In this section we describe our ML methodology which relies on support vector machines (SVM). First, we present the feature vectors considered in our study. Finally, we describe our ML model.

### 3.1. Feature vectors

We considered three sets of vectors, which are described below:

1. **Input Vector**: We considered different parameters available in OpenMP as vector input, such as the number of threads, the loop scheduling policy (static or dynamic), and the chunk size (which defines how many loop iterations will be assigned to each thread at a time).
2. **Hardware Counters Vector:** We used the PAPI library to collect information from hardware counters. We considered the following metrics as the most relevant ones: Last level cache misses, L3 and L2 total cache misses for multicore and manycore respectively (**PAPI_L3_TCM, PAPI_L2_TCM events**), data translation lookaside buffer misses (**PAPI_TLB_DM event**), and total cycles (**PAPI_TOT_CYC event**). For acoustic wave model we use only two hardware counters.
3. **Performance Vector:** The output vector, which uses billions execution time as performance characterization metric.

### 3.2. Machine Learning Model

The proposed ML model is based on SVM, which is a supervised ML approach introduced in [6] and extended to regression problems where support vectors are represented by kernel functions [9]. Our ML model was built on top of three consecutive layers, where output values of a layer are used as input values of the next layer (Figure 1). The input layer contains the configuration values from the input vector. The hidden layer contains the SVMs that take values from the input vector to simulate the behavior of hardware counters. Finally, the output layer contains takes each simulated value from the hidden layer to obtain the corresponding GFLOPS and execution time values.

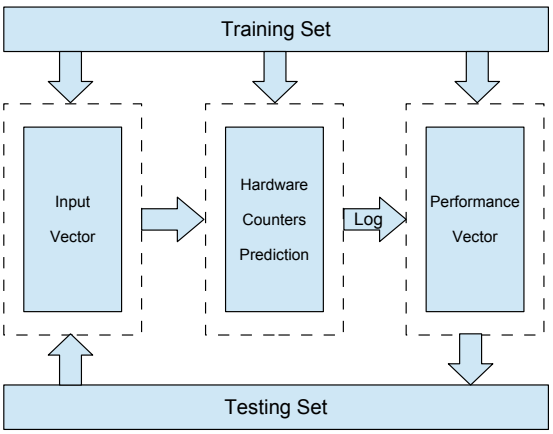



Figure 1: Flowchart of the proposed ML-based model.

## 4. Experiments

In this section we describe our experimental testbed and present the data analysis and results.

### 4.1. Experimental Testbed

We used two multi-core and one many-core platforms to carry out the experiments. Their hardware configurations are shown in Table 1.

| | **Multicore 1** | **Multicore 2** | **Manycore** |
|---|---|---|---|
| *Processor* | Xeon E5-2650 | Xeon E5-4650 | Xeon Phi 7520 |
| *Clock(GHz)* | 2.0 | 2.7 | 1.40 |
| *Cores* | 8 | 8 | 68 |
| *Sockets* | 2 | 4 | 1 |
| *Threads* | 16 | 32 | 272 |
| *Last level cache size (MB)* | 20 (L3) | 20 (L3) | 34 (L2) |

Table 1: Experimental testbed configurations.

**4.1.1. Training and validation sets** We created a training set by randomly selecting a subset from the configuration set presented in Table 2 and details all the configurations available for our optimization categories. As it can be noted, a brute force approach would be unfeasible.

| | **Parameters** | **Multicore 1** | **Multicore 2** | **Manycore** |
|---|---|---|---|---|
| *Number of threads* | 1 | 8 | 12 | 272 |
| *Scheduling policy* | 1 | 2 | 2 | 2 |
| *Chunk size* | 1 | 32 | 32 | 272 |
| *Total of configurations* | 3 | 512 | 768 | 147,968 |

Table 2: Configurations available for our optimization procedure

A random testing set was used since all SVMs in both the hidden and the output layers are trained to calculate new GFLOPS and execution time values through simulation. Table 3 presents the total number of experiments that were performed to obtain the training and validation sets.

| | | **Multicore 1** | **Multicore 2** | **Manycore** |
|---|---|---|---|---|
| *7-point* | Training | 44 | 38 | - |
| | Testing | 11 | 10 | - |
| | *Total* | 55 | 48 | - |
| *Seismic* | Training | 211 | 237 | - |
| | Testing | 53 | 60 | - |
| | *Total* | 264 | 297 | - |
| *Acoustic* | *Training* | - | - | 808 |
| | *Testing* | - | - | 203 |
| | *Total* | - | - | 1,011 |

Table 3: Number of experiments in the training and the testing sets.

### 4.2. Results

We use the validation set to evaluate the model with two statistical estimators: root mean square error (RMSE) and the coefficient of determination (R-square). The former represents the standard deviation of the differences between predicted values and real values whereas the latter represents how close the regression approximates the real data (R-square equal to 1 indicates a perfect fit of data regression).

As it can be noted in Table 4, the RMSE value confirms that deviation of time value is high, but the approximation of R-square is close to 99%, then we get a highly accurate regression.

| | | **Multicore 1** | **Multicore 2** | **Manycore** |
|---|---|---|---|---|
| *7-point* | *RMSE* | 0.66 | 2.51 | - |
| | *R-Square* | 0.98 | 0.86 | - |
| *Seismic* | *RMSE* | 13.53 | 212.28 | - |
| | *R-Square* | 0.99 | 0.62 | - |
| *Acoustic* | *RMSE* | - | - | 154.04 |
| | *R-Square* | - | - | 0.94 |

Table 4: Estimators for predicted values of the numerical kernels.

## 5. Related Works

In [1], the authors focused on acoustic wave propagation equations, choosing the optimization techniques from systematically tuning the algorithm. The usage of collaborative thread blocking, cache blocking, register re-use, vectorization and loop redistribution.

Other works investigated the accuracy of regression models and ML algorithms in different contexts. In [16] the authors compared ML algorithms for characterizing the shared L2 cache behavior of programs on multi-core processors. The results showed that regression models trained on a given L2 cache architecture are reasonably transferable to other L2 cache architectures.

Finally, in [11] the authors applied ML techniques to explore stencil configurations (code transformations, compiler flags, architectural features and optimization parameters). Their approach is able to select a suitable configuration that gives the best execution time and energy consumption. In [8], the authors improved performance of stencil computations by using a model based on cache misses. In [13], the authors proposed a ML model to predict performance of stencil computations on multicore architectures.

## 6. Conclusion

In this paper, we introduced a general predictive performance modeling strategy for geophysical numerical kernel on multi and many-core architectures. We showed that performance of the common numerical kernels can be predicted with a high accuracy (99%) based on hardware counters and Machine Learning.

Firstly, we expect to extend our methodology in order to capture complex behaviors (vectorization capabilities, data mapping). Secondly, we intend to design a model based on unsupervised ML algorithms to further improve our results.

Finally, we believe that a general model can be integrated into an auto-tuning framework to find the best performance configuration for a given stencil kernel.

## 7. Acknowledgments

For computer time, this research partly used the resources of Colfax Research. This work has been granted by CAPES, CNPq, FAPERGS and *PETROBRAS* company. The authors thank Jairo Panetta from Aeronautics Institute Of Technology (ITA) for providing the acoustic wave numerical kernel code. It was also supported by Intel Corporation under the Modern Code Project. Research has received funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E Project, grant agreement n° 689772.

## References

[1] C. Andreolli, P. Thierry, L. Borges, G. Skinner, and C. Yount. Chapter 23 - Characterization and Optimization Methodology Applied to Stencil Computations. In J. Reinders and J. Jeffers, editors, *High Performance Parallelism Pearls*, pages 377 – 396. Morgan Kaufmann, Boston, 2015.

[2] F. Z. Boito, R. V. Kassick, P. O. A. Navaux, and Y. Denneulin. Automatic I/O scheduling algorithm selection for parallel file systems. *Concurrency and Computation: Practice and Experience*, 28(8):2457–2472, 2016. cpe.3606.

[3] M. Christen, O. Schenk, and H. Burkhart. Automatic code generation and tuning for stencil kernels on modern shared memory architectures. *Comput. Sci.*, 26(3-4):205–210, June 2011.

[4] R. G. Clapp. Seismic Processing and the Computer Revolution(s). In *SEG Technical Program Expanded Abstracts 2015*, pages 4832–4837, 2015.

[5] R. G. Clapp, H. Fu, and O. Lindtjorn. Selecting the right hardware for reverse time migration. *The Leading Edge*, 29(1):48–58, 2010.

[6] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[7] K. Datta, S. W. Williams, V. Volkov, J. Carter, L. Oliker, J. Shalf, and K. Yelick. *Scientific Computing with Multicore and Accelerators*, chapter Auto-Tuning Stencil Computations on Multicore and Accelerators. CRC Press, Taylor & Francis Group, 2010.

[8] R. de la Cruz and M. Araya-Polo. *Modeling Stencil Computations on Modern HPC Architectures*, pages 149–171. Springer International Publishing, Cham, 2015.

[9] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. In *Advances in Neural Information Processing Systems 9*, pages 155–161. MIT Press, 1997.

[10] F. Dupros, H. Do, and H. Aochi. On scalability issues of the elastodynamics equations on multicore platforms. In *Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June, 2013*, pages 1226–1234, 2013.

[11] A. S. Ganapathi. *Predicting and Optimizing System Utilization and Performance via Statistical Machine Learning*. PhD thesis, EECS Department, University of California, Berkeley, Dec 2009.

[12] V. Martinez, F. Dupros, M. Castro, H. Aochi, and P. O. A. Navaux. Stencil-based applications tuning for multi-core. In *Latin American High Performance Computing Conference (CARLA 2016)*, pages 1–15, Sep 2016. Oral presentation.

[13] V. Martínez, F. Dupros, M. Castro, and P. Navaux. Performance improvement of stencil computations for multi-core architectures based on machine learning. *Procedia Computer Science*, 108:305 – 314, 2017. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.

[14] R. Mijakovic, M. Firbach, and M. Gerndt. An architecture for flexible auto-tuning: The periscope tuning framework 2.0. In *International Conference on Green High Performance Computing (ICGHPC)*, pages 1–9, Feb 2016.

[15] P. Moczo, J. Robertsson, and L. Eisner. The finite-difference time-domain method for modeling of seismic wave propagation. In *Advances in Wave Propagation in Heterogeneous Media*, volume 48 of *Advances in Geophysics*, chapter 8, pages 421–516. Elsevier - Academic Press, 2007.

[16] J. K. Rai, A. Negi, R. Wankar, and K. D. Nayak. On prediction accuracy of machine learning algorithms for characterizing shared l2 cache behavior of programs on multicore processors. In *Computational Intelligence, Communication Systems and Networks, 2009. CICSYN '09. First International Conference on*, pages 213–219, July 2009.

[17] Y. Tang, R. A. Chowdhury, B. C. Kuszmaul, C.-K. Luk, and C. E. Leiserson. The pochoir stencil compiler. In *ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, pages 117–128, New York, NY, USA, 2011. ACM.

[18] J. Virieux. P-SV wave propagation in heterogeneous media; velocity-stress finite-difference method. *Geophysics*, 51(4):889–901, 1986.

[19] D. Vladuic, A. Cernivec, and B. Slivnik. Improving job scheduling in grid environments with use of simple machine learning methods. In *International Conference on Information Technology: New Generations*, pages 177–182, April 2009.

# Performance Analysis of Machine Learning Algorithms With Different Thread and Data Affinity Approaches

Arthur Mittmann Krause, Eduardo H. M. Cruz,
Matheus S. Serpa, Philippe O. A. Navaux
Universidade Federal do Rio Grande do Sul
Instituto de Informatica
{amkrause, ehmcruz, msserpa, navaux}@inf.ufrgs.br

## Abstract

*The popularity of machine learning algorithms is growing together with their data sets, and so is the need to find ways to accelerate them. Data and Thread mapping is an useful technique to improve performance in those applications. Placing threads that communicate more with each other closer in the memory hierarchy can be beneficial to performance by reducing the latency in the communication, while placing threads that use intensively the same resources in separate cores can reduce competition between them. As a main contribution, we analyze the performance impact of data and thread mappings on popular machine learning algorithms. We managed to reduce the execution time by 87% in some situations using only thread mapping, mostly due to reduction in QPI traffic.*

## 1. Introduction

Machine learning is one of the most rapid growing subjects in computer science, helping nearly all fields of knowledge by bringing a new approach to understanding large data sets. The abundance of training data and techniques currently available are allowing many companies to develop revolutionary products such as self driving cars, personal assistants and precise product recommendations tools. Usually, Machine Learning algorithms are accelerated through cloud services, ASIC or GPU based environments. However, the use of CPU systems can also be interesting because of their flexibility, memory capacity and scalability, and with the new many-core processors, CPUs can be a better choice for largely parallel workloads, which were previously dominated by GPUs.

Threads in multithreaded applications such as Machine Learning algorithms usually share data, forcing it to be moved through the system either by the inter-chip or intra-chip interconnections, depending on the memory hierarchy and the threads location [3]. It means that threads that communicate a lot with each other will experience lower latency from communication if placed closer in the chip. On the other hand, the bandwidth is also shared, which means that threads will compete for it, so threads that are communicating more should be placed far apart from each other, in order to not compete for the interconnections. On NUMA systems, ubiquitous in HPC, data present in the main memory can be closer or farther away, depending on the node it is stored and the physical CPU the thread that is accessing it is running on. Reading data from a distant memory bank results in higher latencies and more traffic in the interconnections.

This paper aims to analyze the performance variations of popular machine learning algorithms, as well as the traffic on the interconnections, when executed using different combinations of thread and data mappings on a NUMA machine, to understand if these applications can be accelerated by a smarter placing of its threads and data and for what reason.

## 2. Related Work

Mazouz et al. analyses the speedup of the SPEC OMP benchmark suite in different machines using various thread affinity strategies. However, no machine learning applications are tested [5]. Bottesch et al. introduces a new method to approximate Euclidian distances in order to significantly accelerate the k-Means algorithm [1]. Morais accelerates the k-Nearest Neighbors algorithm through the use of the Fast Fourier Transform and removing unfavorable training vectors [6]. Both papers provide application specific optimizations, not easily usable in other programs. In this context, this paper aims to analyze the performance implications of different thread and data mappings on popular machine learning algorithms.

## 3. Machine Learning Algorithms

Machine Learning techniques are used to classify elements in large datasets. There are two approaches that are used: supervised and unsupervised [8]. A supervised model aims to classify elements in previously defined categories. In order to achieve this, a previous dataset of values already classified are needed to train the algorithm. This allows the model to know the classes it should attribute to the elements in the data set and the expected features the elements on each class should have. For example, given a table containing the species, length, width, color and quantity of petals of various flowers, a supervised model that has been fed with this data can predict the species of other flowers given their petals characteristics.

Unsupervised approaches are used to group elements in generic clusters. Those algorithms receive some data and cluster the points that have similar features. They are better suited for cases where there is little or nothing known about the dataset and it is wanted to detect patterns within the data. Using the previous example, those models would be fed with just the latter dataset, containing unclassified flowers with the features of their petals, and label them according to what cluster it was found to be part of. The number of clusters can be predefined or variable depending on the technique used. The clusters formed may or may not be equivalent to the original species of the flowers.

In this work, four of the most popular Machine Learning algorithms, two being supervised and the other two unsupervised methods, were used for testing the impact on performance and QPI traffic of thread and data affinity policies. The supervised methods were k-Nearest Neighbors and Back Propagation Neural Network, and the unsupervised were k-Means and Streamcluster. In k-Nearest Neighbors, an element receives the label that is the most frequently occurring between its k-Nearest Neighbors, where k is a number defined beforehand. Back Propagation is an algorithm to train a neural network. The technique is to change the weights of each neuron in the network multiple times in order to minimize the error between the actual output and the desired output of the training set. Once the error is within the tolerable range, the training is complete and the network should be capable of classifying its inputs from outside the training set [7]. In k-Means, a set of k initial centers is generated in the same multidimensional space as the data set one wishes to cluster. The points are then clustered according to their nearest center. New centers are then calculated for each new cluster, and the process is repeated multiple times [4]. The Streamcluster finds a predetermined amount of centers in order to cluster a stream of points in a way that minimizes the sum of the distances between each point and its center. The applications tested are the OpenMP implementations present in the Rodinia Benchmark Suite version 3.1 [2]. We inverted the main loops in k-Nearest Neighbor in order to allow it to run with more than 16 threads.

## 4. Methodology

The applications were executed using different thread and data mappings. The thread affinity was configured through Intel Compiler's OpenMP runtime library predefined mappings. Applications were executed with compact, scatter and no affinity. In compact, the system tries to place the threads as physically close as possible, meaning that the communications between neighbor threads will have less latency, but there will be more competition for resources. With scatter, the threads are placed as far as possible, resulting in more latency from communications, but more resources such as caches, bandwidth and functional units are available. The memory affinity was set using numacontrol. With the interleave policy, memory will be allocated using round robin on nodes, meaning that the data will be spread evenly between the memory banks. With membind, all the data is present in the same node. With no policies set in numacontrol, the default NUMA balance policy of the Linux kernel is active. We also tested the applications with the kernel balancing turned off.

Experiments were done on a machine with four Intel Xeon X7550 processors and 128 GB of DDR3 RAM, being 32 GB for each NUMA node. The applications parameters were selected in order to have a similar execution time and memory footprint. The results shown are based on the mean of six executions. More repetitions were not done because the variance was low and due to time constraints.

## 5. Results

Figure 1 shows the percentage variation of the execution time with different combinations of thread and data mappings, for different amount of threads. Figure 2 shows the percentage variation of the QPI traffic measured using the PCM tool. Each value is calculated from the mean of 6 executions, having as the base point the values for the default thread mapping policy of the Linux kernel and the automatic NUMA memory balancing turned off, for the sake of finding the relative improvement of different mapping combinations in relation to no specific mapping at all. It is possible to observe significant performance improvements with the use of data and thread mappings, specially when using 8, 16 or 32 threads, achieving in some cases more than 87% of reduction on the execution time. In those cases, the thread mapping appeared as the most important factor, while data mapping was less relevant.

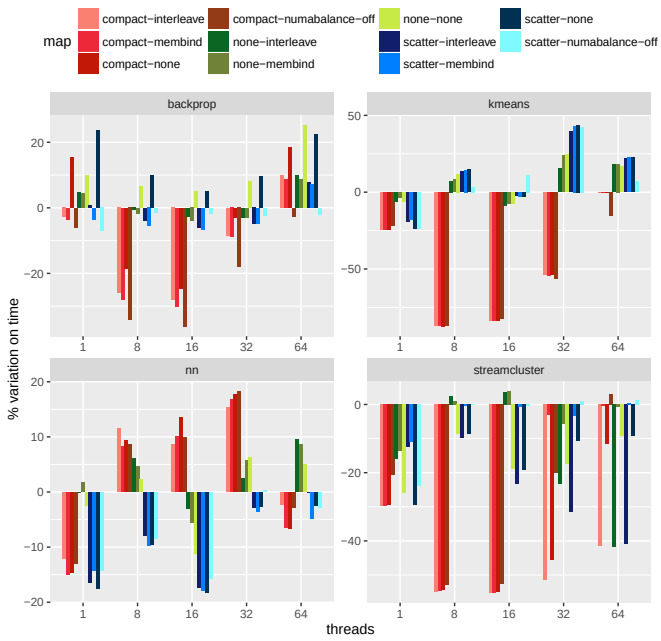



**Figure 1. Percentage variation of the execution time**

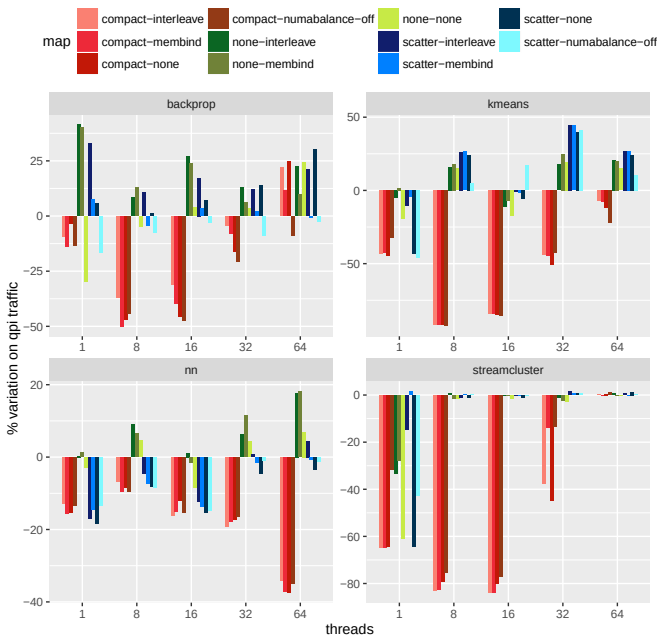



**Figure 2. Percentage variation of the QPI traffic**

The different combinations of thread and data mappings showed different results depending on the application and even thread amount. The compact thread affinity policy yielded the best results on most cases by pinning the threads closer physically and reducing the communication penalties. On k-Means, with 8 and 16 threads, the reduction in execution time was more than 87%, while the QPI traffic was reduced almost equally. A lesser improvement is observed with 32 threads and almost none with 64. Placing the threads closer was also interesting for the Backpropagation with 8, 16 and 32 threads, as well as for the Streamcluster with every thread amount but 64. On the other hand, in the Nearest Neighbors algorithm, where there is little communication between the threads, the scatter policy resulted in better performance, specially with 8 and 16 threads, presumably because there are effectively more physical cores at use, so the competition between resources such as cache lines or functional units was reduced. The similar QPI traffic with compact and scatter mappings on these cases reinforces this conclusion.

Different memory mappings yielded only marginal gains comparad to thread mappings in mosts cases. Nonetheless, With all the 64 cores in use on Streamcluster, the memory interleave policy provided a reduction of more than 40% of the execution time, while the thread mapping policies had no significant impact. There is no reduction in total QPI traffic in this case, because the memory loaded is still the same and the communication between threads is not improved by different mappings when all cores are in use.

As it can be seen by analyzing the Pearson correlation coefficient between the QPI traffic and execution time of the applications when varying the data and thread mappings, as shown in Figure 3, the performance was improved due to the reduction in the QPI traffic in most cases. The exceptions were the Streamcluster with 64 threads, where the QPI traffic didn't change, and the k-Nearest Neighbors, where the performance improvement was due to less sharing of resources and not less traffic on the interconnections.

## 6. Conclusions

Thread and data mappings were found to be effective ways to improve performance of the popular Machine Learning algorithms tested, depending on the applications' communication pattern. We managed to observe an 87% reduction on execution time for k-Means in some cases, and an 82% reduction in QPI traffic for Streamcluster, only with thread mapping. With the interleave memory mapping, reductions as big as 40% on execution time were achieved on Streamcluster when all 64 threads were running. Reductions on execution time were often coupled with similar reductions on QPI traffic. Performance could be further

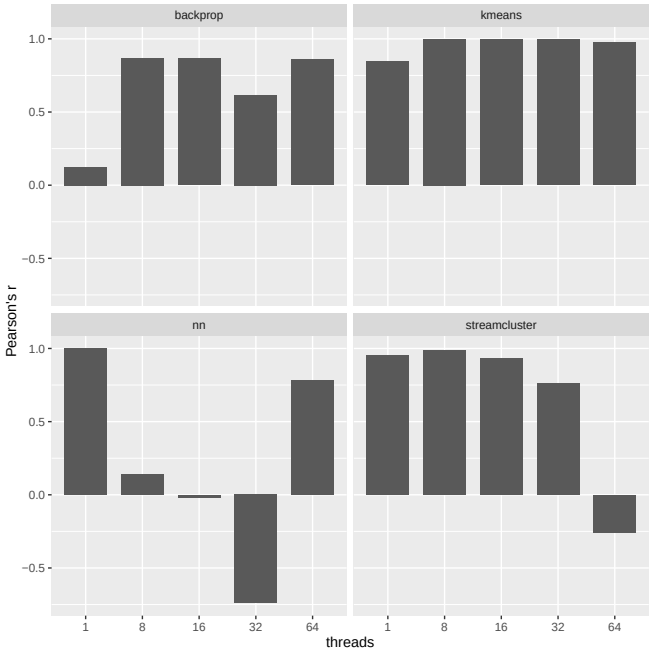



**Figure 3. Pearson's r of the relationship between execution time and QPI traffic**

improved by using more specific data and thread mappings or an online mechanism. Future work will analyze the impact of data and thread affinity on manycore architectures.

## 7. Acknowledgements

This research received funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E project, grant n.o 689772. It was also supported by Intel under the Modern Code Project.

## References

[1] T. Bottesch, T. Bühler, and M. Kächele. Speeding up k-means by approximating euclidean distances via block vectors. In *International Conference on Machine Learning*, pages 2578–2586, 2016.

[2] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, IISWC '09, pages 44–54, Washington, DC, USA, 2009. IEEE Computer Society.

[3] M. Diener, E. H. Cruz, P. O. Navaux, A. Busse, and H.-U. Heiß. Communication-aware process and thread mapping using online communication detection. *Parallel Computing*, 43:43–63, 2015.

[4] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[5] A. Mazouz, D. Barthou, et al. Performance evaluation and analysis of thread pinning strategies on multi-core platforms: Case study of spec omp applications on intel architectures. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 273–279. IEEE, 2011.

[6] J. P. d. Morais Neto et al. Aceleração de uma variação do problema k-nearest neighbors. 2014.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[8] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

# Proposta de balanceamento de carga para a redução do tempo de execução em ambientes multiprocessados *

Vinicius R. S. dos Santos [1], Ana Karina M. Machado [1], Edson L. Padoin[1,2]

[1] Universidade Reg. do Noroeste do Estado do Rio G. do Sul (UNIJUI) – Ijui, RS – Brazil
[2] Universidade Federal do Rio Grande do Sul (UFRGS) – Porto Alegre, RS – Brazil
{vinicius.ribas, ana.morales, padoin}@unijui.edu.br

## Resumo

*Este artigo apresenta uma proposta de um novo balanceador de carga para a redução do tempo de execução de aplicações paralelas executadas em ambientes multiprocessados. O algoritmo do balanceador coleta informações do sistema e da aplicação em tempo real e as utiliza na tomada de decisões de balanceamento de carga dinamicamente, visando reduzir o número de migrações de tarefas enquanto reduzindo o tempo total de execução. Para implementação foi utilizado o modelo de programação paralela* CHARM++. *Os resultados preliminares apresentaram reduções significativas no número de tarefas migradas assim como no tempo total de execução.*

## 1. Introdução

A necessidade de alto desempenho, proveniente de um cresimento da produção de *software* deu abertura para o desenvolvimento dos sistemas computacionais, surgidos para suprir essa demanda.

A partir da evolução do *hardware* do processador, as limitações foram-se sendo ultrapassadas, proporcionando a execução simultâea de diversas aplicações. Isso habilitou à programação pralela a divisão das aplicações em partes inseridas nos núcleos das unidades de processamento [6].

A modelagem de um problema complexo pode culminar em um desbalanceamento de carga e excessiva comunicação entre tarefas [5]. Essa é uma preocupação que surge, devido ao seu caráter impeditivo quanto ao alcance de uma boa eficiência na utilização dos recursos dos sistemas paralelos [4].

Soluções que utilizam estratégias para o emprego adequado dos recursos disponíveis vem sendo desenvolvidas. Balanceadores de Carga(BC), nome dado à estratégia, tem por objetivo detectar e corrigir o desbalanceamento de carga, aprimorando a utilização dos recursos disponíveis a aplicação [3].

O restante deste artigo está assim organizado. A Seção 2 apresenta os trabalhos relacionados. A Seção 4 apresenta a proposta do balanceador. A Seção 3 descreve a metodologia que será empregada na implementação do novo balanceador de carga. Por fim, são discutidos na Seção 6, algumas conclusões e perspectivas de trabalhos futuros.

## 2. Trabalhos Relacionados

Várias plataformas possuem suporte à programação paralela com memória compartilhada e distribuída. Porém, aplicações que foram desenvolvidas utilizando linguagens procedurais, baseadas em um paradigma de troca de mensagem paralelo, apresentam padrões de carga dinâmico, o que tornaria necessária a mudança na estrutura da aplicação [1] para haver um balanceamento de carga.

Diferentes abordagens tem alcançado resultados positivos quando empregado balanceamento de carga para redução do tempo de execução. Dentre elas destacam-se as estratégias centralizadas e distribuídas, sendo atualmente novas abordagem hierárquica vem sendo propostas. Nestas novas abordagens, os núcleos de processamento são divididos em grupos independentes e organizados de uma árvore onde cada nível da árvore é composto por grupos de núcleos. Deste modo, quanto mais núcleos são adicionados aos grupos, menor é o uso da memória pelo BC. Usando esta abordagem Zheng apresenta um BC hierárquica denominado *HybridLB* e con-

* This work was supported by CNPq, CAPES, FAPERGS and FINEP. This research has received funding from the European Community's Seventh Framework Programme (FP7-PEOPLE) under grant agreement number 295217, funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E Project, grant agreement number 689772 and STIC-AmSud/CAPES scientific-technological cooperation program under EnergySFE research project grant 99999.007556/2015-02.

segue *speedup* de 6 com 2.048 *cores* e 145 com 8.192 *cores* [7].

Outros balanceadores abordados no artigo são o RefineLB e o GreedyLB. Ambos são disponibilizados pelo ambiente de programação CHARM++. O BC RefineLB move objetos dos processadores mais sobrecarregados para os menos carregados almejando atingir uma média, sendo limitado o número do objetos migrados [2]. Já o BC GreedyLB possui um algoritmo de abordagem gulosa. Esse paradigma é frequentemente utilizado na teoria e na prática de otimização combinatória. Seu algoritmo objetiva migrar objetos pesados para o processador com menor carga. Isso se repete até que a carga de todos os processadores alcance uma proximidade com a carga média.

## 3. Metodologia

Para validação da proposta, foi utilizado um equipamento com um processador Intel modelo i7-6500U. Este processador possui 4 núcleos com 2 SMT/núcleo, totalizando 8 núcleos. Para os testes, utilizou-se o sistema operacional Linux Ubuntu 16.04 com kernel versão $4.4.33 - 1$. A versão do CHARM++ utilizada foi a 6.5.1 e do compilador g++ a versão 5.4.1.

Os balanceadores de carga foram submetidos a simulações utilizado o *benchmark* LB_Test. Esse *benchmark* foi escolhido por ser facilmente configurável para apresentar diferentes níveis de desbalanceamento de carga, permitindo que a carga computacional de cada tarefa seja configurada em diferentes padrões de carga, tanto de irregularidade quanto de comunicação, além de ser disponibilizado pelo próprio ambiente de programação.

Testes foram realizados com 50 e 100 tarefas, sendo estas com cargas computacionais que variam entre $1500ms$ e 150000ms. As sincronizações para chamada do balanceador de carga foi definidas a cada 10 iterações.

Os resultados alcançados, tempo total de execução e a quantidade total de objetos migrados, foram comparados com os balanceadores de carga REFINELB e GREEDYLB.

## 4. SmartLB

A implementação do balanceador de cargas SMARTLB foi realizada sob o modelo de programação CHARM++, que é um paradigma paralelo voltado à produtividade. Este *framework* de balanceamento de carga foi escolhido uma vez que permite tanto a criação de novos BC quanto a utilização dos BCs disponibilizados pelo ambiente para comparações de resultados.

O CHARM++ é usado para o desenvolvimento de grandes aplicações científicas, possibilitando a execução de programas paralelos com balanceamento de carga automático e *checkpointing* [6].

A estratégia utilizada para implementação do balanceamento de carga proposto, constitui-se de melhorias nas estratégias utilizadas nos algoritmos GREEDYLB e REFINELB. Nossas melhorias buscam equilibrar as cargas entre os processadores reduzindo o número de migrações, adotando um *threshold* para definir o desbalanceamento de carga aceitável.

A partir de informações fornecidas pelo CHARM++, o algoritmo SMARTLB busca atingir balanceamento levando em consideração a diferença de carga entre o núcleo mais carregado e o menos carregado. Migrando tarefas do processador mais carregado para o processador com menor carga, equilibrando a carga total do sistema, reduzindo o número de migrações e o tempo total de execução da aplicação.

Na Tabela 1 são apresentados os principais parâmetros utilizados na implementação do algoritmo proposto.

| Parâmetro | Definição |
|---|---|
| $PM$ | Processador com maior carga |
| $Pm$ | Processador com menor carga |
| $D$ | Desbalanceamento entre $PM$ e $Pm$ |
| $ct$ | Carga da tarefa $P$ |
| $nTarefas$ | Número de tarefas |
| $getProcessadorAtual(i)$ | Retorna o processador atual da tarefa |
| $getCargaTarefa()$ | Retorna a carga da tarefa |
| $getCargaMaior()$ | Retorna a carga do processador mais carregado |
| $getCargaMenor()$ | Retorna a carga do processador menos carregado |
| $migrarProcesso(i, PM, Pm)$ | Migrar tarefa $i$ de $PM$ para $Pm$ |

**Tabela 1. Principais parâmetros utilizados no algoritmo SMARTLB**

Algoritmo 1: Implementação do SMARTLB

```
1  PM = getCargaMaior();
2  Pm = getCargaMenor();
3  if((Pm/PM) > Threshold) {
4      for(i = 1; i <= nTarefas; i++) {
5          if(getProcessadorAtual(i) == PM) {
6              ct = getCargaTarefa(i);
7              D = PM - Pm;
8              if(ct <= D) {
9                  migrarProcesso(i, PM, Pm);
10                 PM = getCargaMaior();
11                 Pm = getCargaMenor();
12             }
13         }
14     }
15 }
```

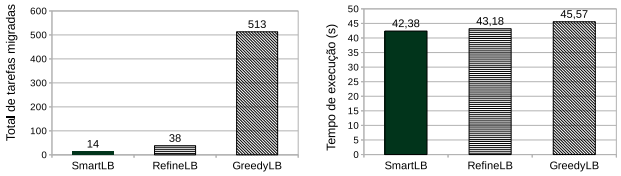
Assim, quando o balanceado é aplicado, ele primeiro analisa a diferença de carga entre o processador mais e menos carregado. Caso essa diferença for maior que o *threshold* o balanceador busca tarefas do processador mais carregado testando se a carga da tarefa é menor ou igual ao

desbalanceamento. Caso seja, ele realiza a migração desta tarefa do processador mais carregado para o menos carregado e encontra o novo processador mais carregado e o novo processador menos carregado, como demonstrado no Algoritmo 1.

Desta forma, consegue-se um balanceamento de carga mais preciso, evitando migrações desnecessárias. Após a execução, quando não existem mais processadores a serem mapeados e as cargas de todas as unidades de processamento possuem um valor próximo um do outro, o balanceamento é encerrado.

## 5. Resultados

Durante os testes, foram levados em consideração o tempo total de execução e a quantidade total de tarefas migradas.

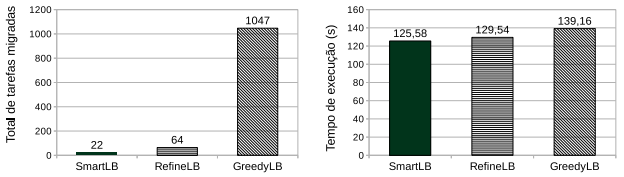



(a) Total de tarefas migradas (b) Tempo total de execução em segundos

**Figura 1. Resultados dos testes com 50 tarefas**

Analisando a Figura 1 Com base nos tempos totais de execução mensurados, observa-se que quando executado os testes com $50$ tarefas, o menor tempo de execução é alcançado pelo algoritmo do SMARTLB. Apresentando um tempo 42,34 segundos, tendo uma diferença de $6,94\%$ menor que o algoritmo do GREEDYLB que por sua vez obteve o maior tempo, com 45,54 segundos.

Observando a quantidade total de tarefas migradas, nota-se quando realizados testes com com $50$ tarefas, que o SMARTLB foi o BC com menor migrações de tarefas, com apenas 14 migrações enquanto o REFINELB e o GREEDYLB migraram 38 e 513 tarefas respectivamente.




(a) Total de otarefas migradas (b) Tempo total de execução em segundos

**Figura 2. Resultados dos testes com 100 tarefas**

Podemos notar na Figura 2, que dobrando a quantidade de tarefas para $100$, mais uma vez o SMARTLB alcançou o menor tempo de execução, bem como também foi o BC que apresentou a menor quantidade de tarefas migradas.

Baseado no tempo total de execução aferido nos gráficos da Figura 2, nota-se que, quando realizados os testes com $100$ processos, o algoritmo do SMARTLB obtém um tempo de execução $3,06\%$ menor que algoritmo do REFINELB. Esse, por sua vez, obteve um tempo $6,91\%$ menor que o BC GREEDYLB, que atingiu o maior tempo, executando seu algoritmo em 139,16 segundos.

Nota-se que aumentando-se o número de tarefas para $100$, o algoritmo do GREEDYLB migra $1047$ tarefas, enquanto os algoritmos do SMARTLB e do REFINELB migram apenas $22$ e $64$ tarefas respectivamente. A disparidade na quantidade de tarefas migradas pelos balanceadores SMARTLB e REFINELB com o GREEDYLB é consequência da sua estratégia gulosa que migra tarefas do processador mais carregado para o menos carregado.

Os algoritmos do RefineLB e do SMARTLB fazem o cálculos para determinar quais tarefas devem ser migrados, evitando desperdícios e, consequentemente, diminuindo a quantidade de tarefas migradas.

## 6. Conclusão

Este artigo apresentou uma proposta de balanceamento de carga para a redução do tempo de execução de aplicações paralelas executadas em ambientes multiprocessados. Os resultados obtidos foram comparados com por outros dois balanceadores de carga.

O balanceador de carga proposto SMARTLB apresentou melhor desempenho nos testes realizados com o benchmark lb_test, tanto na quantidade de tarefas migradas quanto no tempo total de execução quando comparação com os balanceadores de carga REFINELB e GREEDYLB. Estes resultados positivos são alcançados devido ao maior controle na migração das tarefas, evitando assim migrações desnecessárias.

Como futuros trabalhos, pretende-se realizar melhorias no algoritmo SMARTLB almejando gerenciar ainda mais o controle de migrações de tarefas. Pretende-se também realizar testes em sistemas paralelos utilizando problemas reais de computação científica e comparar com outros balanceadores de carga do estado da arte.

## Referências


[1] M. Bhandarkar, L. V. Kalé, E. de Sturler, and J. Hoeflinger. Adaptive load balancing for mpi programs. In *International Conference on Computational Science*, pages 108–117. Springer, 2001.

[2] G. Freytag, G. Arruda, R. S. Martins, and E. L. Padoin. Análise de desempenho da paralelizaç ao do problema de caixeiro viajante. 2015.

[3] B. S. Padilha and E. L. Padoin. Análise de desempenho da aplicação de balanceamento de carga em benchmark sintéticos. *Salão do Conhecimento*, 2(2), 2016.

[4] E. L. Padoin, M. Castro, L. L. Pilla, P. O. Navaux, and J.-F. Méhaut. Saving energy by exploiting residual imbalances on iterative applications. In *High Performance Computing (HiPC), 2014 21st International Conference on*, pages 1–10. IEEE, 2014.

[5] E. L. Padoin, M. B. Castro, L. L. Pilla, T. C. Bozzetti, P. O. A. Navaux, and J.-F. Méhaut. Balanceamento de carga visando reduçao do consumo de energia para o modelo de programaçao charm++. *XIV Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul, Alegrete, RS, Brasil*, 2014.

[6] L. L. Pilla and E. Meneses. teste. *XV Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul, Gramado, RS, Brasil*, 2015.

[7] G. Zheng, E. Meneses, A. Bhatele, and L. V. Kale. Hierarchical load balancing for charm++ applications on large supercomputers. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 436–444. IEEE, 2010.

# Experimental and Analytical Study of Xeon Phi Reliability

Daniel Oliveira
Institute of Informatics, UFRGS
Porto Alegre, RS, Brazil

Laercio Pilla
Department of Informatics and Statistics, UFSC
Florianopolis, SC, Brazil

Nathan DeBardeleben
Los Alamos National Laboratory
Los Alamos, NM, US

Sean Blanchard
Los Alamos National Laboratory
Los Alamos, NM, US

Heather Quinn
Los Alamos National Laboratory
Los Alamos, NM, US

Israel Koren
University of Massachusetts, UMass
Amherst, MA, US

## Abstract

*We present an in-depth analysis of transient faults effects on HPC applications in Intel Xeon Phi processors based on radiation experiments and high-level fault injection. Besides measuring the realistic error rates of Xeon Phi, we quantify Silent Data Corruption (SDCs) by correlating the distribution of corrupted elements in the output to the applications characteristics. We evaluate the benefits of imprecise computing for reducing the programs error rate. For example, for HotSpot a 0.5rate by 85We inject different fault models to analyze the sensitivity of given applications. We show that portions of applications can be graded by different criticalities. For example, faults occurring in the middle of LUD execution, or in the Sort and Tree portions of CLAMR, are more critical than the remaining portions. Mitigation techniques can then be relaxed or hardened based on the criticality of the particular portions.*

# Data mining the memory access stream to detect anomalous application behavior

Francis B. Moreira
Informatics Institute UFRGS
fbmoreira@inf.ufrgs.br

Matthias Diener
Informatics Institute UFRGS
mdiener@inf.ufrgs.br

Philippe O. A. Navaux
Informatics Institute UFRGS
navaux@inf.ufrgs.br

Israel Koren Department of Electrical
Computer
Engineering
University of Massachusetts
koren@ece.umass.edu

## Abstract

*Detecting anomalous application executions is a challenging problem, due to the diversity of anomalies that can occur, such as programming bugs, silent data corruption, or even malicious code corruption. Moreover, the similarity to a regular execution that can occur in these cases, especially in silent data corruption, makes distinction from normal executions difficult. In this paper, we develop a mechanism that can detect such anomalous executions based on changes in the memory access pattern of an application. We analyze memory patterns using a two-level machine learning approach. First, we classify the behavior of different memory access periods within applications using Gaussian mixtures. Then, based on these classifications, we construct matrix representations of Markov chains to obtain information regarding the temporal behavior of these memory accesses. Based on metrics of matrix similarity, we can classify whether the application behaves as expected or anomalously. Using gradient boosting on the metrics of matrix similarity, our technique correctly classifies more than 85of all executions, identifying instances of the same application and different applications. We can also detect a range of faulty executions caused by benign or malicious permanent bit flips in the code section.*

# List of Authors