BSPonP2P: Towards Running Bulk-Synchronous Parallel Applications on P2P Desktop Grids

Rodrigo da Rosa Righi, Gustavo Rostirolla, Vinicius Facco Rodrigues, Alexandre Veith, Cristiano André da Costa
Applied Computing Graduate Program, Unisinos, Av. Unisinos, 950, São Leopoldo, Rio Grande do Sul, Brazil

Abstract— *Today, BSP (Bulk-Synchronous Parallel) represents one of the most used models for writing tightly-coupled parallel programs. A BSP application is divided in one or more supersteps, each one ending with a synchronization barrier. As resource substrates, commonly clusters and eventually computational grids are used to run BSP applications. In this context, we investigate the use of collaborative computing and idle resources to execute this kind of demand, so we are proposing a model named BSPonP2P to answer the following question: How can we develop an efficient and viable model to run BSP applications in P2P Desktop Grids? We answer it by providing both process rescheduling and checkpointing, enabling BSPonP2P to address dynamism at application and infrastructure levels and resource heterogeneity. The results concern a prototype that ran over a subset of the Grid5000 infrastructure, showing encouraging results on using collaboration and volatile resources for obtaining High Performance Computing effortlessly.*

Keywords: Bulk-Synchronous Parallel, P2P, Process Rescheduling, Checkpointing, Performance

1. Introduction

The widespread use of parallel machines crucially depends on the availability of a model of computation simple enough to provide a convenient basis for software development. Concerning this, the Bulk Synchronous Parallel (BSP) model of computation has been proposed by L. G. Valiant as an unified framework for the design and programming of general purpose parallel computing systems [17]. BSP applications are composed by a set of processes that execute supersteps, each one divided into three phases: (i) local computations on each process; (ii) global communication actions and; (iii) a synchronization barrier. The barrier phase should wait for the slowest process before starting the next superstep, so an efficient process-processors mapping is crucial for getting an acceptable performance [8]. This topic is yet more relevant when considering heterogeneous (different processing and network bandwidth capacities) and dynamic (fluctuations in network bandwidth and processors' load) environments [12].

BSP represents a common used model for writing successful parallel programs that exhibit phase-based computational behaviors, being extensively used to organize MPI (Message Passing Interface) codes [6]. As deployment machines, this programming model has been used on clusters and computational grids [7]. Particularly, this kind of grid is known by normally presenting a centralized or hierarchical architecture, high-speed networks linked to the Internet and nodes that slowly change their participation behavior along the time [5], [7]. In addition, for using one of the aforementioned parallel machines, the user must either buy the computational and network infrastructures or present a previous contract/agreement with the institution that host them. Concerning this landscape, we started the study of low cost and collaborative environments to take profit of end nodes around the Internet effortlessly. This effort culminated in an architecture proposed by Zhao, Liu and Li named P2P Desktop Grids [19] (PDG). Although joining the power of idle resources, a high level of dynamism with the sudden leaving of users (and consequently, reducing also the available CPU cycles of the system) and the use of worldwide-scale and Internet-based connections are challenges when associating this architecture with the purpose of HPC (High Performance Computing). In this way, our work presents the following problem statement: *How can we explore collaborative computing in PDG to run BSP applications efficiently?*

Aiming at answering the posed question, we are proposing *BSPonP2P* - a model that encompasses an infrastructure, overlay network, scheduling algorithms and runtime management to run BSP programs in PDG. Unlike computational grids, the target architecture is not managed by skilled professionals. BSPonP2P addresses collaborative computing at middleware level, where programmers do not need to change their applications in order to execute them in a P2P setting. Taking profit of the formal organization of BSP programs and mixing structured and nonstructured P2P deployments, the proposed model addresses performance but do not putting away the need of a fault tolerance layer that is crucial on PDG. This article describes BSPonP2P and its runtime strategies to answer the problem statement. Moreover, we also present its evaluation with a BSP scientific application in a real Grid infrastructure. In the best of our knowledge, BSPonP2P is the first proposal to cover BSP programs and P2P settings, being the pioneer on covering round-based parallel applications with spatial decoupling in an easier and costless way.

The remainder of this article will first introduce the related work in Section 2. Section 3 describes BSPonP2P

in details, demonstrating its rationales and contributions. Evaluation methodology and the discussion of the results are presented in Sections 4 and 5, respectively. Finally, Section 6 emphasizes the scientific contribution of the work and notes several challenges that we can address in the future.

2. Related Work

This section briefly presents initiatives to run applications on collaborative environments. Focusing to support Bag-of-Tasks (BoT) applications, the authors in [1] present a generic content-based publish/subscribe system called DPS. Moreover, Leite et al. [10] propose a load balancing architecture using a P2P-like structure for desktop grids.

Besides BoT, the master-slave approach is addressed in [4], [14], [15], [18]. Balasubramaniam et al. [2] and Byung et al. [16] presented aproaches targeting Desktop Grids, and Godfrey et al. [4], Shudo [15], Senís et al. [14] and Wu and Tian [18] present aproaches targeting PDG.

Concerning BSP parallel applications, both Mizan [8] and Camargo et al. [3] are representative for heterogeneous and dynamic environments. Mizan is a dynamic load balancing that captures data from computation and communication metrics. Camargo enable the use of not only idle processor cycles, but also unused disk space of shared machines, and a checkpointing-based mechanism.

Table 1 presents a summary of the aforementioned systems and algorithms. As shown, the initiatives approach different models for collaborative environments and assorted scheduling strategies. We can note that few works are focusing on metrics different of computation, as well as on failure control. In this regard, we observe a research opportunity to work with tightly-coupled applications, such as BSP, on collaborative environments, offering pertinent strategies to cover BSP features on highly dynamic and heterogeneous substrates.

3. BSPonP2P: Proposal to Run BSP Programs in P2P Desktop Grids

Considering both the Internet advances and the ease on acquiring computing resources (in this case, personal computers, tablets and smartphones), we observe the increase adoption of collaborative computing to run any kind of applications at a low financial cost. The simple idea is to profit idle CPU cycles, since the aforementioned resources are used mostly to access social networks, Internet queries and programs that consume a low computational substrate [9].

In this regard, we are proposing the BSPonP2P model to design how BSP applications would run in PDG efficiently. To accomplish this, BSPonP2P proposes a network overlay architecture, as well as strategies to turn viable the matching involving collaborative infrastructure and the BSP programming model. To accomplish this viability, we are exploring both process checkpointing and rescheduling. Checkpointing brings reliability and performance saving to the model: when someone leaves the system in a superstep then a checkpoint is used to restart the application in the last saved point. Rescheduling, in its turn, aims at covering dynamism, since both nodes and networks can become overloaded at application runtime; so, process can be on-the-fly migrated to novel locations to improve application performance. Particularly, rescheduling is highly pertinent in BSP programs, since they are composed by supersteps limited by a synchronization barrier in which the slowest process always bounds the parallel performance.

3.1 Network Overlay Architecture

BSPonP2P architecture was developed taking in mind both structured and unstructured P2P networks, as shown in Figure 1. Firstly, we are working with a structured ring-based network following the so-called Chord P2P protocol [11]. Chord uses a DHT and a Finger table to provide message exchange and routing in an efficient, scalable and secure way. This kind of network is used to connect nodes defined as Managers. We are using a timer denoted tto (time-to-organize) to reorganize the Managers in the Chord ring, aiming at optimizing communication latency among them. Each Manager is responsible for a specific cluster, where the cluster here means a parallel machine, a local network, a mobile device, or a single computer. Structured P2P networks aim at providing performance for large scale deployments [19], emphasizing our design decision for using them for Managers Interactions because of we plan to compose a worldwide scale architecture. A cluster, in turn, is organized in an unstructured manner. This decision was taken because this kind of organization offers better flexibility and dynamism with heterogeneous and unstable resources. The nodes inside a cluster are named End Nodes, or only Nodes, and they are responsible to execute the BSP applications actually.

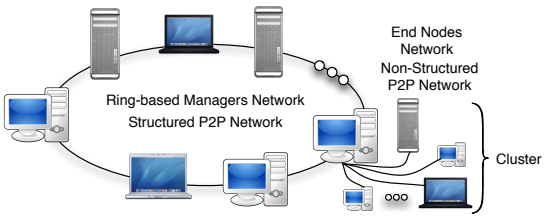

Fig. 1: Computational Overlay Network with two communication levels: (i) among the Managers; (ii) between a Manager and an End Node.

Each resource can act as a Manager or End Node. We created a Computational Overlay Network (CON) to manage message routing, scheduling, as well as the entrance and the leaving of a resource in the infrastructure. Each cluster i has a maximum of n_i End Nodes. This value is configurable and the administrator can change it according to network size. Thus, the first resource will act as a Manager and the others

Table 1: Comparison among initiatives to run parallel applications on collaborative environments.

Initiatives	Target system	Model application	Migration	Data Replication	Load balancing	Monitoring
Anceaume et al. [1]	PDG	Bags of Task	-	-	Computation	Computation
Balasubramaniam et al. [2]	Desktop Grids	Master/Slave	-	-	Computation	Computation
Camargo et al. [3]	PDG	BSP	yes	yes	Computation	Computation
Godfrey et al. [4]	PDG	Master/Slave	yes	yes	Computation	Computation
Khayyat et al. [8]	Desktop Grids	BSP	yes	no	Computation	Computation
Leite et al. [10]	PDG	Bags of Task	yes	yes	Work Stealing	Computation
Sentís et al. [14]	PDG	Master/Slave	yes	no	Computation	Computation and Memory
Shudo et al. [15]	PDG	Master/Slave	yes	no	Computation	Computation
Byung et al. [16]	Desktop Grids	Master/Slave	yes	no	Computation	Computation
Wu et al. [18]	PDG	Master/Slave	yes	yes	Computation	Computation

up to n_i will serve as End Nodes to the composed cluster i. After reaching n_i, another Manager is selected and then, other cluster is created. Upon entering the network, an End Node must report how much of their computing resource will be available to BSPonP2P. The resource's owner sets this parameter. By default, 100% of CPU is available when the user is not using the equipment or a percentage is employed, otherwise.

Aiming at getting End Node data periodically, a Manager sends query requests at intervals of t_i seconds. Each Manager defines t_i for cluster i at launching time. The queries are sended in a random Walk strategy. By definition, if the Manager does not receive a response from an End Node two consecutive times, then the Manager disconnects it from the CON. This procedure is executed to ensure that the Node is connected and able to execute a process from a future demand. Each cluster with less than n End Nodes is a candidate to receive the next incoming resource. Among them, the cluster with the lowest identification is actually chosen to host this new resource.

CON automatically reorganizes the network when a node, either a Manager or End Node, suffers a crash or intentionally leaves the collaborative infrastructure. In the case of a Manager, the oldest End Node in the cluster (comparing the stay time in the CON, and not other metrics like computational power since a Manager is responsible mainly for routing) is promoted to be the Manager. This new Manager is then updated with the data about cluster itself, supersteps in execution (if any) and the applications running in the resources under its responsibility. If an End Node crashes and it was executing supersteps of one or more applications, the Manager has partial data about the execution and can select other peer in accordance with the scheduling function to relaunch the application from the last saved checkpoint.

3.2 Scheduling and Runtime Strategies

This subsection describes how are deploying a BSP application in a P2P setting, showing also the runtime strategies to address performance and fault tolerance. We are targeting phases-based applications such as BSP, however the application model applied in BSPonP2P diverges from the traditional BSP [17], since this last one was firstly defined for homogeneous clusters.

The communication within the CON is divided into two levels, depending on the node's role: the first level comprises communication among the Managers, while the second level represents an interaction between a Manager and an End Node. After this introduction, the application launching occurs as follows. An End Node has a BSP demand and submits it to its Manager (the Manager just acts as organizer, it doesn't process the demand), informing the binary code and the number of process to run the application. Using the first-level communication, the mentioned Manager chooses the target cluster for each process.

The second-level communication is important to notify a Manager to choose an End Node under its responsibility to run a process. After selecting one End Node per process, an Execution Network is composed as depicted in Figure 2. It comprises a direct connection of each End Node (computing resource) to the Manager that started the BSP demand. This Manager will coordinate process communication and will pass the final result to the requester. The idea here is to save hops while performing communication actions among the BSP processes.

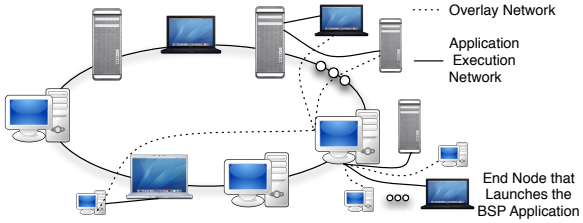

Fig. 2: Creation of an Execution Network, in which the Manager that receives the BSP demand acts as a gateway that directly communicates with all End Nodes involved in the BSP computation.

The evaluation of the first level will decide which cluster will execute a particular process. For that, we are using a decision function denoted PM (Potential of Migration) proposed by Righi et. al. [12] in the MigBSP proposal. PM is computed through Equation 1, which receives as inputs i and j, a process and a cluster, respectively. In this context, $Comp$, $Comm$ and Mem denote computation, communication and memory metrics, respectively. The larger the PM value, the most profitable is the target cluster j in receiving a process i. Each process is tested against each cluster and the largest PM value indicates the cluster for the process. Different from MigBSP, BSPonP2P uses PDG,

which implies in a modification of the computation metric in accordance with Equation 2. $T(i)$ and $Set(j)$ are inherited from MigBSP [12], and denote the computational time of process i in the last superstep and the relative performance of the cluster j, respectively. BSPonP2P adds $\overline{X}_{Resource}$ and $\overline{X}_{User}$ with the following objectives:

- **User**: here, $\overline{X}_{User(j)}$ is used to evaluate the users that either are running or have already ran applications in the resources of the cluster j in evaluation. If the users in the aforementioned context present a behavior of low usage of CPU power, this metric is close to 1. On the other hand, the value is close to 0 if the historical data does demonstrate a higher utilization of CPU cycles.
- **Resource**: The metric $\overline{X}_{Resource(j)}$ denotes an arithmetic average of the resources utilization in the cluster j. The idea is to work with a historical data in this metric, where close to 1 represents a lower usage of CPU or close to 0, otherwise. Thus, User metric analyses the pattern of access by the users, while the Resource observes the utilization degree of the resources.

$$PM(i,j) = Comp(i,j) + Comm(i,j) - Mem(i,j) \quad (1)$$

$$Comp(i,j) = (\frac{\overline{X}_{Resource(j)} + \overline{X}_{User(j)}}{2}).T(i).Set(j) \quad (2)$$

Table 2 presents an example of how we are computing both User and Resouce metrics. The first column of this table refers to the captured observations on each cluster. In this scenario there are three users, named X, Y and Z, and two clusters, $C1$ and $C2$. $C1$ is composed by machines the A and B, while $C2$ includes machines C and D. Assuming the distribution among the clusters as presented in Table 2, the computation of the metric Users in cluster $C2$ will result 25% ($\frac{10\%+20\%+30\%+40\%}{4}$) while the Resource usage in cluster $C2$ will result in 15% ($\frac{10+20}{2}$). Note that the user Z does not have influnce in the User metric for cluster C2, because it is not performing any task in cluster C2. In the same way, Resource for C1 is 53.33% ($\frac{30\%+40\%+90\%}{3}$), while User for this cluster is 38% ($\frac{10\%+20\%+30\%+40\%+90\%}{5}$).

Table 2: Example of infrastructure usage, including 2 clusters and 3 users

Observation	User	Machine	Cluster	% of Machine use (CPU)
1	X	C	C2	10
1	Y	D	C2	20
2	X	A	C1	30
2	Y	A	C1	40
2	Z	B	C1	90

The evaluation with the second communication level is used to define which End Node in a cluster will run a specific process. The definition of the executor node is made based on the availability of the equipment. At this point, a simple assessment is made, where samples of at least three ratings and a maximum of ten reviews of availability (amount of computational resource available) are used. The samples are based on past records received by the Manager.

As runtime strategies, BSPonP2P offers process rescheduling and checkpointing. Both take place after ending a particular superstep, this point refers to a consistent global state of the distributed system. The idea is to offer a runtime management that aims at reducing the load imbalance among the processes, so decreasing the execution time of each superstep. Rescheduling tests are done not at each superstep, but the superstep index is defined in accordance with the MigBSP parameter called α. The system is launched with a predetermined value of α, which represents the interval of supersteps to evaluate process rescheduling. At each α supersteps rescheduling may occur if there is another most suitable cluster to execute a process according to PM function.

Aiming at dealing with dynamic environments, BSPonP2P profits from the phases-based organization of a BSP application to take a distributed snapshot of the application. This is done by saving a local checkpoint in each process, representing a basic BSPonP2P mechanism for addressing fault tolerance. The idea consists of not restarting the application from scratch in the presence of a node failure or outgoing user. Only data of the last superstep is saved, since all processes advance in a round-based fashion. This feature allows a time reduction if happens any crash in the system, for example, if anyone that is participating in a superstep leave the network (a Manager or End Node), the model have been projected to restart from the last point saved. The execution only lose a few supersteps and it doesn't need to restart from the beginning of the demand.

3.3 Observing Different Scenarios and Goals

The BSPonP2P's differential approach is highlighted by the adoption of process migration and checkpointing. Figure 3 illustrates different scenarios after running a BSP application using BSPonP2P. Scenario i represents the simple execution of a BSP application, disabling any service or scheduling functionality. The End Node-process mapping is fixed, being defined by the user.

Scenario ii adds the scheduling calculus in the first and second levels of the CON. Scenario i always outperforms scenario ii, since this last one adds dynamic scheduling overhead. Situations c, d, e and f represent the possibilities found in scenario iii. This scenario enables process checkpointing and rescheduling. Situations c and d present not suitable migrations, or yet, migrations were profitable but the number of remaining supersteps are not so large to get back the time in migration investment.

Both situations e and f represent the success in running BSPonP2P. Although situation e has a larger time when compared to situation a, it was computed using the check-

pointing strategy. Therefore, we can gain by not needing to restart the application from the first superstep in the case of a node crash. Process migration was responsible for a better resource usage and performance in the last situation.

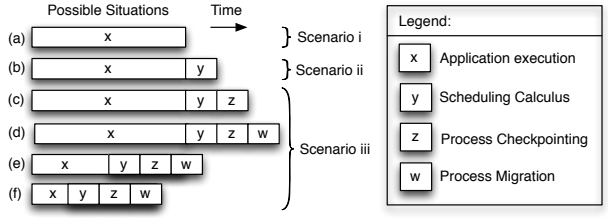

Fig. 3: Different execution scenarios of BSPonP2P. Both situations e and f of scenario iii are considered the main BSPonP2P's goal.

4. Evaluation Methodology

The evaluation was performed using SimGrid[1], a deterministic scientific instrument to study the behavior of scheduling algorithms in heterogeneous platforms, due to its high adoption in the scientific community. We applied simulation in three different scenarios using the Simgrid's MSG module: (i) Simple application execution; (ii) Application execution along with BSPonP2P scheduler without migration or checkpoint; (iii) Application execution with BSPonP2P scheduler with migration and checkpoint. The scenarios are graphically presented in Figure 3. The objective of the mentioned scenarios is to show the overload imposed by BPSonP2P (comparing scenarios i and ii), and the gain or loss of time when migration is enabled (comparing scenarios i and iii).

We also performed a recovery validation: in this case, we evaluated the time to resume the execution with checkpoint and compared with the time without this service. Without the checkpoint when a fault occurs, the system restarts from the beginning of the execution, instead of resuming from the last saved barrier as explained before. The comparison will be based on the exit of a host running a process in BSPonP2P.

We implemented a BSP application for computational fluid dynamics based on the principle of the Lattice Boltzmann Method (LBM) [13]. Each superstep is modeled by dividing the data in blocks, where each process performs a local computation using the block and, after that, sends updated data to its neighbor at the right. In order to perform the tests we allocated the first 15 nodes from each of the following Grid5000 clusters[2]: chimint and chicon located in Lille, paradent from Rennes, grephene from Nancy, gdx from Orsay, capricorne from Lyon, adonis from Grenoble, borderplage from Bordeaux, pastel from Toulouse and suno from Sophia, giving a total amount of 150 nodes.

[1]http://simgrid.gforge.inria.fr

[2]Details about computing resources and network connections can be found at http://www.grid5000.fr

Tests conducted in each scenario suffered the variation of three parameters: (i) α, which defines the interval of supersteps to perform the migration process starting with 4, 8 and 16 (same values used by [12]); (ii) Amount of supersteps whose values tested were 10, 50, 100, 500, 1000 and 2000; (iii) Amount of processes, assuming the values 11, 26, 51 and 89, randomly chosen to represent the environment that is found in PDG.

In order to represent the user interaction with the nodes and the variation of cluster availability we also created weight vector which represents the amount of computation available for the process to be executed. This values varied between 30 and 99 percent during all the executions changing at each superstep.

5. Discussing the Results

This section presents the results obtained when executing the LBM in the PDG varying the α value, number of supersteps and processes against each scenario. First we are going to evaluate BSPonP2P in the aforementioned scenarios without any machine leaving the CON. After, we evaluate the variation in the average time of supersteps and finally the checkpoint activation impact in the execution time when a machine leaves the CON.

5.1 Analysis of the Model Parameters

Analyzing the changes in the execution time according to the variation of the parameters presented above we can observe that the load imposed by BSPonP2P checkpointing and migration varies between an improvement of 6% with 26 processes, 2000 superstep and α equal to 16 and an overload of 17% with 89 processes, 100 supersteps and α equal to 4. Yet, in 76% of the cases the overload imposed was smaller than 5% in the total execution time. Also with α equal to 4 scenario iii is always better than scenario ii and with α equal to 8 scenario iii is better than scenario ii in all the cases with more than 11 processes.

The variation in execution time with different α values is better observed in Figure 4 where a comparison of the execution time between scenarios i and iii is presented. With the number of supersteps above 500 there is a decrease in the execution time, varying between -2.9% and -4.5% when α is equal to 4, -1.6% and -4.5% when α is equal to 8, and -3.8% to -5.5% when α is 16.

When evaluating the average time between each superstep, presented in Figure 5, we can observe a variation between 23.4 seconds with α 4 and 20.1 seconds with α 16. In some cases the migration do not present an improvement in the execution time, nevertheless with all α values there is an improvement in the average time between supersteps if compared to it's initial value (i.e. after the first migration).

The increase found in migration 3 and 4 in Figure 5 with α equal to 8 is given by the increased use of resources. The cluster elected to run the migration 4 had PM equal 2.89 and

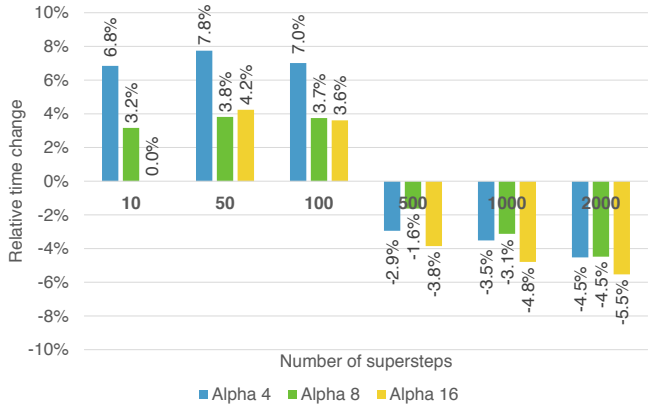

Fig. 4: Relative time variation of scenario iii when compared to scenario i varying the number of supersteps with 26 processes.

the average use of the cluster was 15% (18% of use among users and 12% utilization of the equipment). In the step that the migration 4 occurred the cluster PM which was running the previous superstep was 2.72. The reduction comparing the PMs was generated from the historical consumption of 18% found in the cluster that was running step 3, in this context the process was migrated to higher PM. Thus, the small increases found in Figure 5 are generated by differences in consumption of the user equipment and the network participants that occurred after the migration.

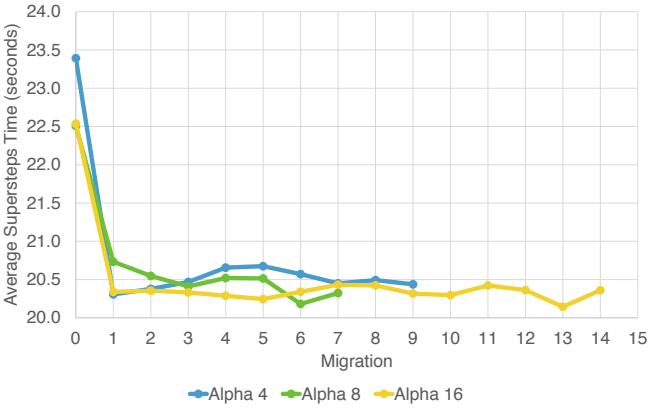

Fig. 5: Average time of supersteps in each migration with 26 processes varying the α value.

This dynamism in the resource utilization generated by the participants of the network leads to a tendency of unbalanced tasks. In Figure 6 we can observe that despite of alpha and amount of processes variation the migrations occurred along the entire execution.

This variation is also confirmed observing Figure 7 that shows the final tasks allocation with 51 processes and all α values. Despite of better computational resources of cluster Graphene (144 CPUs Xeon X3440, 16 GB memory and Infiniband-20G) when compared to Chicon (52 CPUs

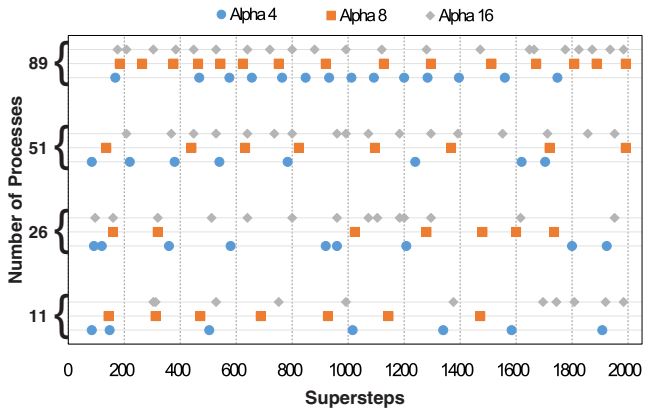

Fig. 6: Migrations distribution along the application execution varying the α value and number of processes.

Opteron 285, 4 GB memory and Myri-10G) for instance no migration pattern to this cluster can be detected. This behavior is highly related to Equation 2 which represents the computation resources usage by the user and equipments and is also used in migration calculus.

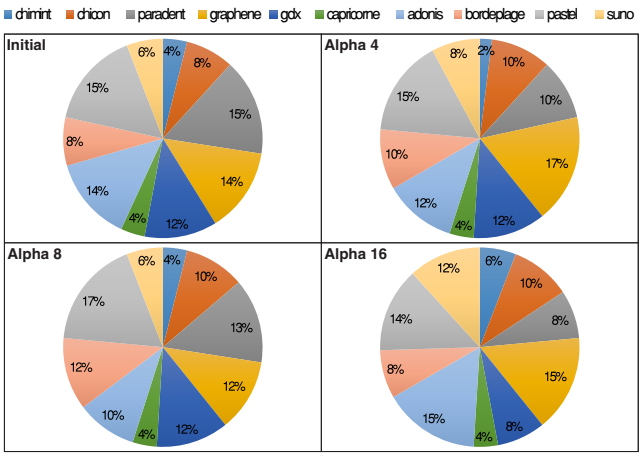

Fig. 7: Distribution of 51 processes among the clusters. The first graph indicates the initial distribution and the others the final distribution according to the α values.

5.2 Impact of the Checkpoint Activation

As mentioned earlier, we evaluated the overhead caused by BSPonP2P calculus, and the migrations occurred during an execution without any machine leaving the CON. In this experiment we analyze the recovery after an unexpected exit of a machine from the CON, since a user can leave the P2P network anytime. When there is no checkpoint the application must restart the execution from beginning, *i. e.*, scenario i, although with BSPonP2P whenever a migration occurs a checkpoint is saved and the application can restart from this point. In this context we evaluate scenario iii with 89 processes running, 2000 supersteps and α equal to 16, simulating an exit in different supersteps as can be seen in Figure 8.

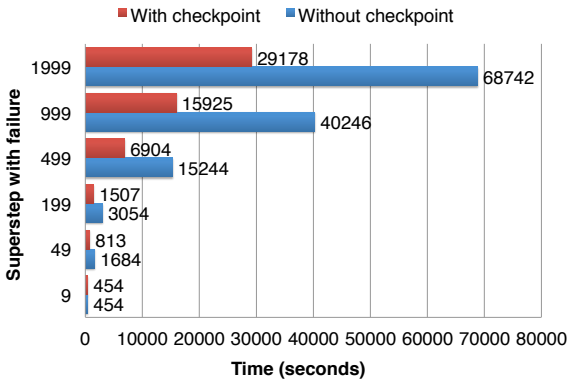

Fig. 8: Performance with and without checkpointing according to the supersteps with failure

Observing the results we can see that an exit in superstep 9 for instance did not cause any gain because there was no migration and consequently no checkpoint. On the other hand, when there is a bigger amount of superstep and a closer checkpoint, like the one occurred in the last case with an error in the superstep 1999 and the last checkpoint in the superstep 1016, an economy of more than 57% in time could be obtained.

6. Conclusion

This article presented BSPonP2P as an alternative to run BSP applications in PDG infrastructures. To the best of our knowledge, the proposed model is the first that joins the aforementioned programming model and the collaborative execution environment. Process rescheduling and checkpointing management is the BSPonP2P's scientific contribution. Thanks to both strategies, we demonstrated that the word "efficiency" referred in the problem statement means here performance and fault tolerance.

Besides presenting situations in which BSPonP2P outperforms the simple execution of a BSP application, most of the results using Grid5000 clusters showed an average overhead of 1.09% when using process rescheduling and checkpointing. We classify this rate as positive to BSPonP2P, because of an application must not be restarted from the scratch when any fault occurs (either when a node crashes or when an user sudden leaves the collaborative infrastructure).

Finally, we would like to emphasize that BSPonP2P is not restricted to BSP applications, but it can be used to manage any round-based computations in collaborative environments. Future research should evaluate BSPonP2P with process replication in order to launch copies of a process at specific superstep to run concurrently, so helping at both performance and fault tolerance areas.

Acknowledgment

This paper was partially founded by Santander and the following Brazilian agencies: CNPq, CAPES, FAPERGS.

References

[1] E. Anceaume, M. Gradinariu, A. Datta, G. Simon, and A. Virgillito. A semantic overlay for self- peer-to-peer publish/subscribe. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE Int. Conf. on*, pages 22–22, 2006.

[2] M. Balasubramaniam, N. Sukhija, F. Ciorba, I. Banicescu, and S. Srivastava. Towards the scalability of dynamic loop scheduling techniques via discrete event simulation. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th Int.*, pages 1343–1351, 2012.

[3] R. Camargo, F. Castor, and F. Kon. Reliable management of checkpointing and application data in opportunistic grids. *Journal of the Brazilian Comp Society*, 16(3):177–190, 2010.

[4] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in dynamic structured p2p systems. In *INFOCOM 2004. Twenty-third AnnualJoint Conf. of the IEEE Comp and Communications Societies*, volume 4, pages 2253–2262 vol.4, March 2004.

[5] F. P. Hargreaves, D. Merkle, and P. Schneider-Kamp. Group communication patterns for high performance computing in scala. In *Proceedings of the 3rd ACM SIGPLAN Workshop on Functional High-performance Computing*, FHPC '14, pages 75–85, New York, NY, USA, 2014. ACM.

[6] B. Hendrickson. Computational science: Emerging opportunities and challenges. *Journal of Physics: Conf. Series*, 180(1):012013, 2009.

[7] K. Khan, K. Qureshi, and M. Abd-El-Barr. An efficient grid scheduling strategy for data parallel applications. *The Journal of Supercomputing*, 68(3):1487–1502, 2014.

[8] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis. Mizan: a system for dynamic load balancing in large-scale graph processing. In *Proceedings of the 8th ACM European Conf. on Comp Systems*, EuroSys '13, pages 169–182, New York, NY, USA, 2013. ACM.

[9] E. Kijsipongse and S. U-ruekolan. Scaling hpc clusters with volunteer computing for data intensive applications. In *Comp Science and Software Engineering (JCSSE), 2013 10th Int. Joint Conf. on*, pages 138–142, May 2013.

[10] A. F. Leite, H. C. Mendes, L. Weigang, A. C. M. A. Melo, and A. Boukerche. An architecture for p2p bag-of-tasks execution with multiple task allocation policies in desktop grids. *Cluster Computing*, 15(4):351–361, 2012.

[11] L. Lin, K. Koyanagi, T. Tsuchiya, T. Miyosawa, and H. Hirose. Improving routing load balance on chord. In *Advanced Communication Technology (ICACT), 2014 16th Int. Conf. on*, pages 733–738, Feb 2014.

[12] R. d. R. Righi, L. Graebin, and C. A. da Costa. On the replacement of objects from round-based applications over heterogeneous environments. *Software: Practice and Experience*, pages n/a–n/a, 2014.

[13] C. Schepke and N. Maillard. Performance improvement of the parallel lattice boltzmann method through blocked data distributions. In *Comp Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th Int. Symposium on*, pages 71–78, Oct 2007.

[14] J. Sentís, F. Solsona, D. Castellà, and J. Rius. Discop2p: an efficient p2p computing overlay. *The Journal of Supercomputing*, 68(2):557–573, 2014.

[15] K. Shudo, Y. Tanaka, and S. Sekiguchi. P3: P2p-based middleware enabling transfer and aggregation of computational resources. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE Int. Symposium on*, volume 1, pages 259–266 Vol. 1, 2005.

[16] B. H. Son, S. woo Lee, and H.-Y. Youn. Prediction-based dynamic load balancing using agent migration for multi-agent system. In *High Performance Computing and Communications (HPCC), 2010 12th IEEE Int. Conf. on*, pages 485–490, 2010.

[17] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, Aug. 1990.

[18] D. Wu, Y. Tian, and K.-W. Ng. On the effectiveness of migration-based load balancing strategies in dht systems. In *Comp Communications and Networks, 2006. ICCCN 2006. Proceedings.15th Int. Conf. on*, pages 405–410, 2006.

[19] H. Zhao, X. Liu, and X. Li. A taxonomy of peer-to-peer desktop grid paradigms. *Cluster Computing*, 14(2):129–144, 2011.