# Evaluating Parameter Sweep Workflows in High Performance Computing

Fernando Chirigati, Vitor Silva, Eduardo Ogasawara, Jonas Dias, Fabio Porto, Patrick Valduriez, Marta Mattoso

# Evaluating Parameter Sweep Workflows in High Performance Computing[*]

Fernando Chirigati[1,#], Vítor Silva[1], Eduardo Ogasawara[1,2], Daniel de Oliveira[1],
Jonas Dias[1], Fábio Porto[3], Patrick Valduriez[4] Marta Mattoso[1]
[1]COPPE/Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
[2]CEFET/RJ, Rio de Janeiro, Brazil
[3]LNCC, Rio de Janeiro, Brazil
[4]INRIA & LIRMM, Montpellier, France
{fernando_seabra, silva, ogasawara, danielc, jonasdias, marta}@cos.ufrj.br
fporto@lncc.br
Patrick.Valduriez@inria.fr

## ABSTRACT

Scientific experiments based on computer simulations can be defined, executed and monitored using Scientific Workflow Management Systems (SWfMS). Several SWfMS are available, each with a different goal and a different engine. Due to the exploratory analysis, scientists need to run parameter sweep (PS) workflows, which are workflows that are invoked repeatedly using different input data. These workflows generate a large amount of tasks that are submitted to High Performance Computing (HPC) environments. Different execution models for a workflow may have significant differences in performance in HPC. However, selecting the best execution model for a given workflow is difficult due to the existence of many characteristics of the workflow that may affect the parallel execution. We developed a study to show performance impacts of using different execution models in running PS workflows in HPC. Our study contributes by presenting a characterization of PS workflow patterns (the basis for many existing scientific workflows) and its behavior under different execution models in HPC. We evaluated four execution models to run workflows in parallel. Our study measures the performance behavior of small, large and complex workflows among the evaluated execution models. The results can be used as a guideline to select the best model for a given scientific workflow execution in HPC. Our evaluation may also serve as a basis for workflow designers to analyze the expected behavior of an HPC workflow engine based on the characteristics of PS workflows.

## Categories and Subject Descriptors

H.4.1 [**Information Systems Applications**]: Workflow management. H.2.8 [**Database Management**]: Scientific. I.6.7 [**Simulation and Modeling**]: Simulation Support Systems Environments

## General Terms

Measurement, Performance, Experimentation.

## Keywords

Scientific Workflows, Execution Model, Parameter Sweep, High Performance Computing (HPC)

## 1. INTRODUCTION

Many scientific experiments are based on computer simulations. There is now a growing use of scientific workflows to organize and manage these experiments. Workflows can be characterized as models of processes, composed of activities and their dependencies [1]. Scientific workflows are data-centric workflows that represent scientific experiments as graphs, where nodes correspond to activities and edges correspond to data being passed between activities. Activities invoke scientific programs that prepare, process and analyze scientific data.

Scientific workflows are typically defined, executed and monitored by Scientific Workflow Management Systems (SWfMS) [2, 3]. Currently, several SWfMS are available, *e.g.* Taverna [4], Kepler [5], VisTrails [6], Pegasus [7], and Swift [8]. Each SWfMS supports its own language for expressing workflows and it is backed by a well-defined execution model. As a result of such heterogeneity, the execution of a scientific workflow presents distinguishable behaviors when run by different SWfMS.

An important characteristic of scientific workflows is their focus on data intensive processing. A typical example is the iterative evaluation of a workflow over different input parameter values, known as *parameter sweep* [9]. In some scenarios, the space of parameter values may be as large as thousands of elements. In this context, in order to produce results in a reasonable time, a High Performance Computing (HPC) environment is needed. Some SWfMS, such as Pegasus, Swift and Chiron, provide support for HPC environments [10]. Others, due to restrictions on their execution engines, need to be combined with a specialized HPC middleware, *e.g.* MapReduce [11] and Hydra [12].

All of these parallel execution engines need to choose the best model for a given scientific workflow execution in HPC. When it comes to data-intensive execution on HPC environments, more aspects, such as data transfer methods, have to be taken into account by the workflow engine.

*SWEET 2012, May 20, 2012, Arizona, Texas, USA.*

[*] Work partially sponsored by CAPES, CNPq and INRIA (Datluge and Sarava projects).

[#] Currently at Polytechnic Institute of NYU.

A typical distributed or parallel database query optimizer considers different combinations of algorithms and data transfer methods before designing the execution plan [13]. For example, when optimizing the execution of two joins, it can ship the entire relation produced by the first join to the next or the second join can fetch tuples as needed and the two joins will execute in a parallel pipeline model.

In a similar way, the workflow execution engine optimizer can choose the data transfer method, among several, from waiting for all data to be generated by the first activity to be entirely sent to the next activity, or to starting the second activity as the first generates its partial results in a pipeline execution model. This pipeline depends on the constraints of the application, but it is often possible to choose between different options. Kepler allows for the workflow designer to choose the execution model while selecting a director for the workflow. However, choosing between the data transfer execution models is not simple for the workflow designer in data-intensive workflows. Even when the choice is made by the optimizer of the workflow execution engine, it is difficult to predict the performance behavior. If this decision is static, i.e. made before the execution begins, and fixed for the whole workflow it may lead to poor performance.

To address this problem, we developed a study to measure and compare the performance of different execution models in running parameter sweep (PS) workflows. As PS workflows may follow different patterns [14] and workflow activities may have different characteristics and execution times, our study defines five PS workflow patterns. These PS workflow patterns are evaluated using metrics that are focused on workflow execution time. Besides the design of the PS workflow patterns, we have also developed a data generator tool to fully recreate PS workflow scenarios. It creates input data for all PS workflow patterns based on two independent variables that can be used to characterize the size and complexity of the scientific experiment.

Our experimental study aims to evaluate the behavior of PS workflows on four different execution models we implemented. These models are related to parallel tasks dispatching strategies over computing resources. It is possible to make an analogy between each evaluated execution model and classical data-transfer methods such as MapReduce or pipeline execution models. In our experimental evaluation, we were able to measure performance differences of distinct execution models to execute variations of scientific workflows. We were also able to measure the scalability of the evaluated execution models.

The paper is organized as follows. Section 2 gives some background on the different targets of computational environments. In Section 3, a characterization of scientific workflow activities is presented in detail. Section 4 describes the design of our study. Section 5 shows the experimental evaluation. Section 6 presents related work. Section 7 concludes the paper.

## 2. COMPUTATIONAL ENVIRONMENT

When we evaluate PS workflows in HPC, there are different characteristics of the computational environment that need to be taken into account. The performance of a workflow execution model in HPC may vary significantly under different environments, which may lead to unfair comparisons. Concerning our research, we are interested in HPC environments, such as clusters, grids [15] and clouds [16]. However, in this paper, we focus on cluster systems with distributed memory and shared disk storage – a very common architecture in many research centers.

Clusters are usually a set of homogeneous computing nodes connected through a high speed and low latency network. On a distributed memory cluster, each processor has its own private memory. The communication between processes is done through message passing such as MPI [17]. Too much communication may present an overhead. However, in the PS workflow scenario, each process (or task) executes an instance of a workflow activity using a given combination of parameters. Thus, the processes usually do not share or exchange data, so there would be no communication between processes. In contrast, on a shared memory system, processes share a single memory space. They can take advantage of the single memory space to communicate, reading and writing shared parameters in the memory. PS workflow cannot take too much advantage of shared memory system because their processes usually do not exchange data.

PS workflow activities can be distributed through a scheduler and executed in parallel by several nodes of a cluster to speed up execution time. Distributed memory systems need very efficient scheduling and dispatching mechanisms to improve task distribution [18] that should take advantage of stable and homogeneous characteristics of the cluster to improve the execution performance. Data storage and access may also impact the performance of the execution model when executing scientific workflows. Considering the data storage options, clusters can deploy either shared-nothing or shared-disk architectures [19]. A shared-nothing architecture relies on exclusive access to local disks. The data is split through the environment nodes that can only access its data partition. In shared-disk, all nodes share the same disk space.

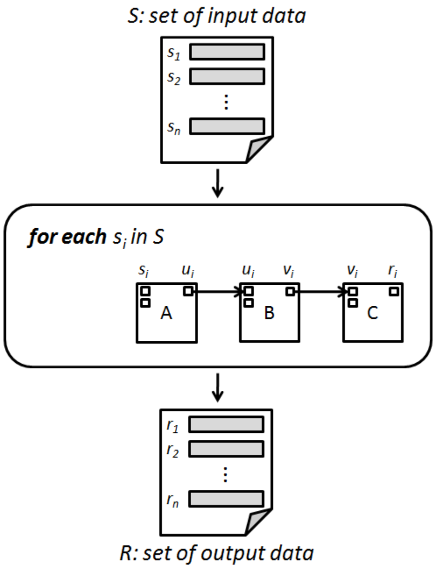


**Figure 1. PS workflows**

## 3. CHARACTERIZATION OF PS WORKFLOWS

A scientific workflow is a data-centric workflow composed of a set of activities. Activities execute programs that consume and produce data (parameters values and files). An output data produced by an activity can be consumed as input data to another activity, establishing a dependency relation between those activities. The workflow engine executes workflow activities respecting their dependency order. During the execution of an experiment, scientists explore the behavior of their workflow model under different inputs. A common problem in scientific experiments is to run a scientific workflow many times as a parameter sweep. In these scenarios, we usually have a set of input data, which may be consumed and transformed by many activities to produce a final set of output data for the entire experiment. Consequently, each activity of a workflow is executed repeatedly by consuming elements of the set of the input data to produce elements of the set of output data. Figure 1 shows an example of a PS workflow representation, where *S* is the set of input data consumed by the workflow, *R* is the produced set of output data, and $s_i$ and $r_i$ are, respectively, elements of the set of input data and elements of the set of output data. Activity A consumes a particular $s_i$, produces data $u_i$ that is consumed by activity B, and activity B produces data $v_i$ that is consumed by activity C to produce $r_i$.

As PS workflows run activities repeatedly, it is important to characterize activities by the way in which they consume and produce data [21, 22]. In this scenario, activities can be interpreted as functions that transform data according to a particular ratio of consumption/production [21, 22]. Under this criterion, we consider four basic types of activities (Map, Split Map, Reduce and Join). Activities that execute a *Map* function (Figure 2a) are the ones that produce a single output data for each input data consumed, *i.e.* with an input/output ratio of 1:1. Activities that execute a *Split Map* function (Figure 2b) are the ones that produces a set of output data for each input data, *i.e.* with an input/output ratio of 1:n. Activities that execute *Reduce* function (Figure 2c) are the ones that produce a single output data from a set of input data, *i.e.* with an input/output ratio of n:1. Finally, activities that execute Join function (Figure 2d) are the ones that consume a set of data that may come from a fixed set of *m* different activities to produce a single set of output data, *i.e.* with an input/output ratio of n:n.

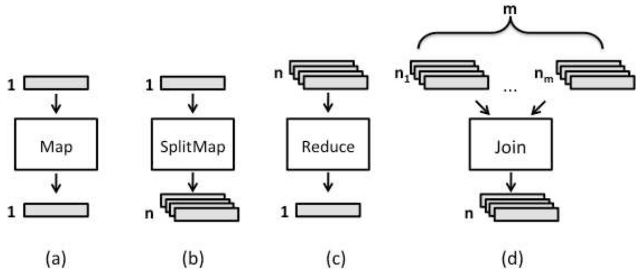


**Figure 2. Activities with data consumption and production: *Map* (a); *Split Map* (b); *Reduce* (c); *Join* (d)**

Furthermore, it is important to characterize patterns that are presented in many PS workflows, *i.e.*, characterize the way in which the aforementioned types of activities are commonly combined. Although we do not intend to entirely cover all possible PS scientific workflows, it is possible to observe five PS workflow patterns (Sequence, Thread Split, Thread Merge, Parallel Split and Synchronization) that are present in many real PS workflows, such as in Provenance Challenges Workflows [22], Computational Fluid Dynamics (CFD) [12], Bioinformatics [24, 26], ultra-deep-water oil exploitation [21], algorithmic skeletons [24] and characterization papers [21, 23]. These PS workflow patterns are related to previously characterized workflow patterns [14] and we use the same notation as described in this work.

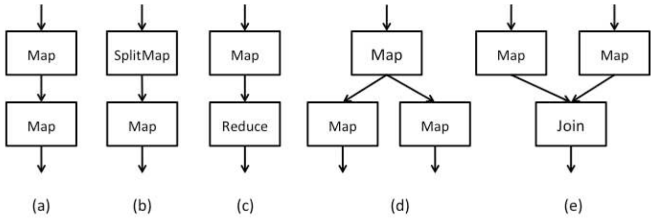


**Figure 3. Workflow patterns mapped in PS workflows: WP-1 (a); WP-42 (b); WP-41(c); WP-2(d); WP-3(e)**

The Sequence pattern (WP-1) (Figure 3a) corresponds to a sequence of Map activities, *i.e.*, a Map activity is enabled after the completion of the preceding Map activity. The Thread Split pattern (WP-42) (Figure 3b) corresponds to a sequence of SplitMap and Map activities. The input data for the SplitMap activity generates a set of output data. Each element of this output data is individually consumed by the Map activity further down the workflow. The Thread Merge pattern (WP-41) (Figure 3c) corresponds to a sequence of Map and Reduce activities. When previous Maps produce all output data, the Reduce activity is then executed, consuming all input data at once and producing its single output data. Another relevant pattern is the Parallel Split pattern (WP-2) (Figure 3d), which corresponds to a broadcast of the output of a particular Map to *n* different following Map activities, *i.e.*, the following *n* activities receives a copy of the same output data, and these activities may execute concurrently. Finally, in the Synchronization pattern (WP-3) (Figure 3e), the input data comes from *n* different types of Map activities and joined to produce a set of output data. The synchronization pattern requires synchronism, *i.e.* the Join activity only executes when its set of input data is ready to be consumed.

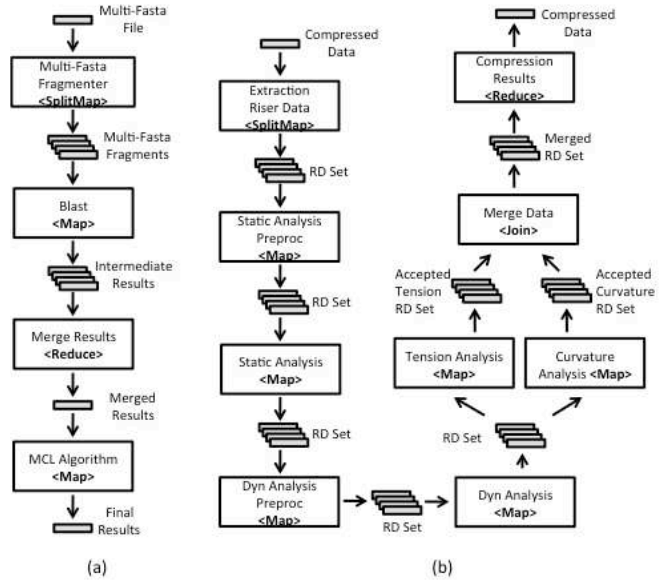


**Figure 4. OrthoMCL (a) and RFA (b) workflows**

In Figure 4, we illustrate two real PS workflows characterized and composed according to the type of activity (Map, SplitMap, Reduce, Join). The OrthoMCL workflow (Figure 4a) [23] provides a scalable method for identifying orthologous groups regarding genome evolution. The Risers Fatigue Analysis workflow, or simply RFA workflow (Figure 4b) [21], aims at computing the fatigue of risers, tubular structures that are used to pump oil from ultra-deep-water of the ocean to the surface. In both workflows, it is possible to observe the presence of the proposed workflow patterns and type of activities.

# 4. EVALUATION STUDY DESIGN

Workflow engine designers may want to know which execution model provides the best performance for executing their scientific workflow in a particular computational environment. Typically, a PS workflow is repeatedly executed by exploring different combinations of input data (parameters and files). Therefore, choosing an adequate execution model may lead to an economy in scale. The workflow execution performance may considerably vary according to different parallelization strategies and different computational environments. The goal of our study is to establish a reference for performance comparison between four different execution models for PS workflow execution. Besides, it aims to be reproducible so that other workflow optimizers may adequate the study to their models of execution.

## 4.1 Workload Configuration

We introduce the workflow workload configuration by discussing the independent variables of our study, named *scaling factors*. The scaling factors allow us to customize the workload of our PS workflow patterns to make it representative of different types of scientific experiments. Scaling factors are represented as integer values that characterize the complexity of activities and the size of input parameter space. The parameter space involves the parameters sets to be explored in the parameter sweep like shown on Figure **5**. In our study, parameter sets are identified by a key parameter $k_i$ and each parameter set is consumed by an instance of a given activity *A*, for example. The parameter sweep involves the execution of the same activity for each parameter set. Changes in scaling factors influence the cardinality of the parameter space and the parameter values that specifies the duration of the activity. Thus, the factors may significantly affect the performance of PS workflow execution. Table 1 shows the description of scaling factors.

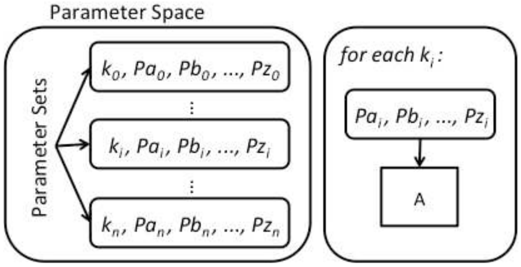

Parameter Space
Parameter Sets
$k_0$, $Pa_0$, $Pb_0$, ..., $Pz_0$
⋮
$k_i$, $Pa_i$, $Pb_i$, ..., $Pz_i$
⋮
$k_n$, $Pa_n$, $Pb_n$, ..., $Pz_n$
for each $k_i$:
$Pa_i$, $Pb_i$, ..., $Pz_i$
A

**Figure 5. Parameter space consumption by an activity of a workflow**

ISF is the scaling factor responsible for defining the number of times the workflow is executed using different input data, *i.e.*, the cardinality of the input parameter space. The cardinality is defined as $2^{\text{ISF}+8}$, for ISF ≥ 1. This arbitrary definition creates parameter spaces with at least 512 ($2^9$) input parameter sets. ACF is a positive integer value that is related to the duration (in seconds) of the activities. The duration of each activity follows a Gamma distribution $\Gamma(\kappa, \theta)$, where $\kappa = 2^{\text{ACF}}$ and $\theta = 1$, for ACF $\geq 1$. Gamma distribution is commonly used to represent expected queue time, which makes it convenient to represent also the duration of activities.

While ISF is used to generate the right amount of data for the input parameter space for each PS workflow pattern, the ACF is used to specify the duration of the activities.

**Table 1. The scaling factors used in the study.**

| Variable | Description |
|---|---|
| Instance Scale Factor (ISF) | Integer value that defines the cardinality of the input parameter space for the PS workflow. |
| Activity Cost Factor (ACF) | Integer value that defines a basis to specify the duration of the activity execution. |

## 4.2 PS Workflow Patterns

Representative PS workflow patterns can be executed with different types of activity. They correspond to building blocks in PS workflows, which, once executed, stress the workflow engine.

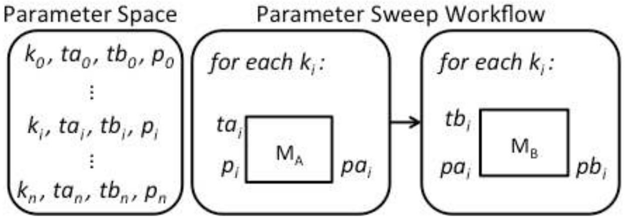


**Figure 6. Map PS workflow pattern**

We designed five PS workflow patterns that refer directly to the five patterns defined in Section 3. We used four different activities: M, S, R, and J. PS workflow patterns are designed as combinations of these activities. Activity M is related to the *Map* function, consuming a single input data to produce a single output data. The S type is related to the *Split Map* function and fragments the input data into several output data. The R type is related to *Reduce* function and aggregates several input data into a single output data. Activity J is related to the Join function. Our PS workflow patterns focus on activities that are data-centric and computational intensive.

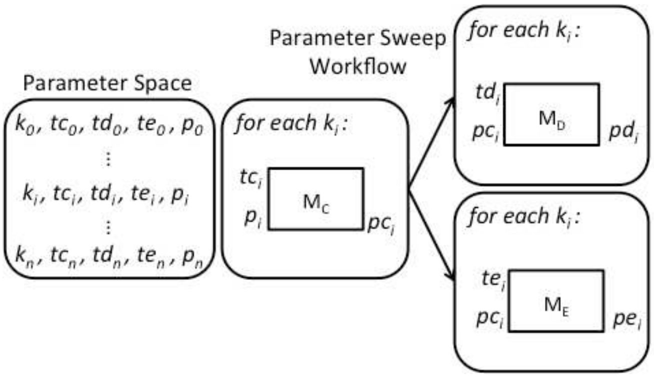


**Figure 7. Broadcast PS workflow pattern**

*Map* PS workflow pattern (Figure 6) represents a sequential flow of activities, based on the *WP-1* pattern (Figure 3a). It is a fundamental building block for workflows [14] and can be found in several of them [20], thus becoming an essential part to be considered by our study. It defines activities that are sequenced, *i.e.*, one activity is executed once the previous one has been completed. This flow of activities is executed for each input parameter set identified by $k_i$ in the parameter space. In Figure 6, each input parameter set identified by $k_i$ contains the parameters $ta_i$ and $tb_i$ that indicates the duration of the activities $M_A$ *and* $M_B$, respectively, related to the ACF value. The remaining parameter *p* represents a random value that is consumed by activity $M_A$, which produces an output *pa* that is later consumed by $M_B$.

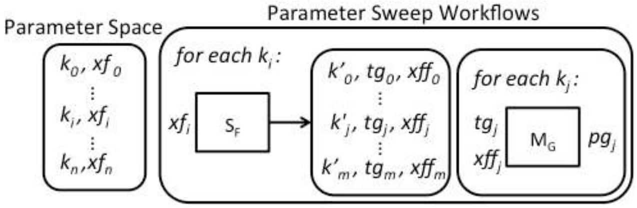


**Figure 8. Split Map PS workflow pattern**

*Broadcast* (Figure 7) and *Split Map* (Figure 8) PS workflow patterns are based on the *WP-2* (Figure 3d) and the *WP-42* (Figure 3b) patterns, respectively. They are important as they increase the number of data that need to be processed in PS workflows [20], which is particularly significant when considering HPC environments. In Figure 7, each output data produced by activity $M_C$ is broadcasted, *i.e.*, replicated in branches to the following activities: $M_D$ and $M_E$. Again, the parameters *tc*, *td* and *te* are related to ACF and indicate the duration for activities $M_C$, $M_D$ and $M_E$, respectively. The input parameter space also contains the input random parameter *p* that is consumed by activity $M_C$ to create the output *pc*, which is later broadcasted. In Figure 8, each parameter value *xf* of activity $S_F$ (outer *for each*) is decomposed, or fragmented, into multiple smaller data ($x_{ff}$), which are consumed by activity $M_G$ (inner *for each*). Activity $S_F$ also produces the parameter *tg* that indicates the duration for the activity $M_G$ based on the chosen ACF value.

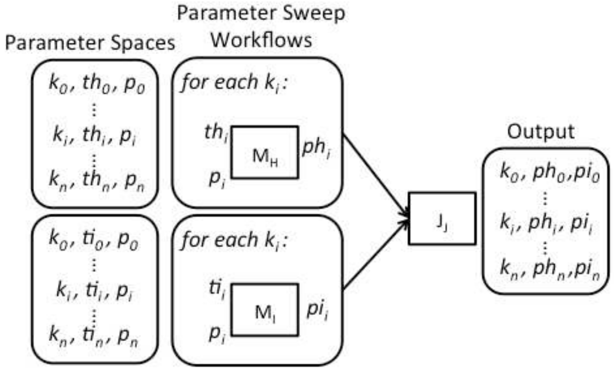


**Figure 9. Join PS workflow pattern**

*Join* (Figure 9) and *Reduce* (Figure 10) PS workflow patterns are based on the *WP-3* pattern (Figure 3e) and the *WP-41* (Figure 3c) respectively. In the *Join* PS workflow pattern, activities $M_H$ and $M_I$ are executed for each parameter set in their respective input parameter space. Activity $J_J$ is responsible for combining their sets of output data into a single set according to a given criteria. Differently from the *Join* PS workflow pattern, in *Reduce* PS workflow pattern, multiple output data ($pl_i$) of the same activity $M_L$ are combined into a single output data in reduce activity $R_M$. In other words, a single output data in $R_M$ is produced for each subset of $M_L$'s output data. These PS workflow patterns are important because they may represent a reduction in the number of data that need to be processed by PS workflows [20].

The proposed PS workflow patterns specify the design of the workflows in our evaluation study, using the scaling factors defined in the workload configuration to setup their execution. ACF is used to specify the duration of the activities, given by the parameters starting with *t* in Figure 6 to Figure 10. Besides, the ISF factor is used to generate the right amount of data for the input parameter space for each PS workflow pattern.

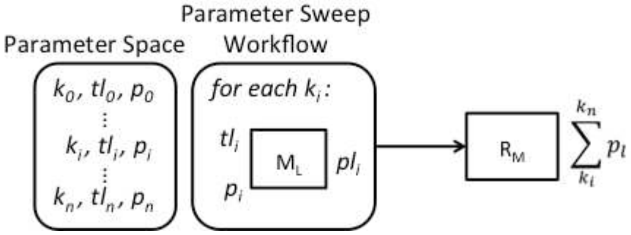
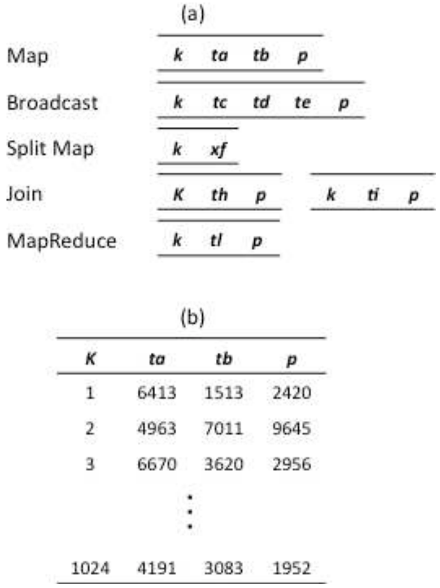


**Figure 10. Reduce PS workflow pattern**

## 4.3 Running PS Workflow Patterns

After configuring the workload, *i.e.* the desired values for ACF and ISF, two more steps are necessary to run the PS workflow patterns: generating input data using the data generator tool and executing, in fact, the PS workflow patterns. Each PS workflow pattern involves running a particular PS workflow. These PS workflow patterns require a particular set of input data to produce a set of output data. The dataset generator program is responsible for creating all necessary data to execute the parameter sweep execution.

In order to generate the set of input data, it is necessary to specify the required PS workflow pattern and the values of the scaling factors (ISF and ACF). Figure 11(a) shows the input parameters generated by the program for each PS workflow pattern. For example, the input parameters generated for the Map PS workflow pattern are *k*, *ta*, *tb* and *p*. Additionally, Figure 11(b) shows an example of the generated set of input data for the Map PS workflow pattern, considering ISF=2 and ACF=1. The set of input data, *i.e.*, the parameter space file is generated as a CSV (comma separated values) file.



(a)

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Map | *k* | *ta* | *tb* | *p* | | | | |
| Broadcast | *k* | *tc* | *td* | *te* | *p* | | | |
| Split Map | *k* | *xf* | | | | | | |
| Join | *K* | *th* | *p* | | *k* | *ti* | *p* | |
| MapReduce | *k* | *tl* | *p* | | | | | |

(b)

| *K* | *ta* | *tb* | *p* |
|---|---|---|---|
| 1 | 6413 | 1513 | 2420 |
| 2 | 4963 | 7011 | 9645 |
| 3 | 6670 | 3620 | 2956 |
| ⋮ | | | |
| 1024 | 4191 | 3083 | 1952 |

**Figure 11. Input parameters for the PS workflow patterns cases (a) and an example for Map (b)**

After generating the input data, running a PS workflow pattern includes the execution of activities (M, S, J or R) using the correct parameters. To simplify the command line execution, each activity of the PS workflow patterns can be invoked using a particular Java program. Since there are four activity types, we developed four Java programs named *M.jar*, *S.jar*, *J.jar* and *R.jar*. They require a small fingerprint to be used (more specifically, the ability to run Java programs). The syntax to invoke the activities programs varies according to the type of activity (M, S, J or R) and also includes their parameters values. Table 2 shows the input and output parameters for each program and their command line syntax.

**Table 2. Command lines to invoke the activity programs used in the study**

| Name | Input | Output | Command line |
|---|---|---|---|
| M.jar | $k, t, p_i$ | $p_o$ | java –jar A.jar -K=$k$ -T=$t$ -P=$p_i$ |
| S.jar | $k, x_f$ | $x_{ff}$ | java  -jar S.jar -K=$k$ -X=$x_f$ |
| J.jar | $j_a, j_b$ | $j_o$ | java -jar J.jar -J=$j_a$,$j_b$ |
| R.jar | $r_i$ | $v$ | java -jar R.jar  -R=$r_i$ |

The activity program M.jar receives three input parameters. The $k$ parameter is the key of the parameter set in the parameter space. The $t$ parameter is the duration of the activity associated to the ACF factor and finally $p_i$ is the random value that flows across the workflow. The activity program S.jar receives only the key parameter $k$ and the file to be fragmented indicated by parameter $x_f$. The activity program J.jar receives two files as input parameters $j_a$ and $j_b$ and creates a new file $j_o$. The activity J is characterized as a n:n data consumption/production ratio function, thus it consumes two datasets to join them on a single parameter space using their keys to combine them. As previously mentioned, the parameter spaces are stored as CSV files, thus activity J consumes two CSV files to produce a single one. The activity program R.jar receives the input files $r_i$ to be reduced to the single output value $v$. The program R.jar executes a sum of all values of $p$ found inside the input file $r_i$.

Depending on the execution model being evaluated, it may be interesting to use a native Join or Reduce function to perform these operations instead of using activity programs J.jar or R.jar. Using a native function would give more accurate and fair results for the given model. For example, when using Hadoop, scientists can program Hadoop Reduce function to execute activity program R instead of using R.jar.

## 4.4 Performance Metrics

We divide performance metrics into four metrics, all of them related to the execution time of PS workflows. Table 3 presents metrics used in our evaluations.

The two main metrics for comparison are Elapsed Time ($T_E$) and Score ($S_E$). $T_E$ is the elapsed time to run the workflow using the input data. Score is computed as the sum of the Efficiency (E) result from each PS workflow pattern. The score aims to be a simple value that can be used to compare different execution models. Table 4 shows the formulas for each metric. The Speedup (S) is computed as a division between the time of the sequential execution ($T_1$) and the time of the parallel execution ($T_E$).

The efficiency E is computed as S/$p$, where $p$ is the number of cores. The score metric, $S_E$, is computed as the average of the results for efficiency $E_x$ for each PS workflow pattern $x$. The speedup, the efficiency and the score are metrics used to measure the performance for PS workflow patterns. In this way, the metric $S_E$ supports the comparison between different execution models, considering all PS workflow patterns.

Although we propose these given metrics related to execution time, we believe that a broader study regarding the best metrics to evaluate PS workflows, similar to what was done is [26], is necessary. We leave this for future work.

**Table 3. Evaluation metrics used in the study**

| Metric | Description |
|---|---|
| Elapsed Time ($T_E$) | Total workflow execution time (in minutes). |
| Speedup (S) | Value that corresponds to the equivalent sequential execution time over the parallel execution time. |
| Efficiency (E) | Value between 0 and 1 that represents how efficient the available cores are used to execute the workflow. |
| Score ($S_E$) | Metric used to define the performance of the execution model considering all PS workflow patterns. |

**Table 4. Metrics and their formulas used in the study**

| Metric | Formula |
|---|---|
| **Speedup (S)** | $S = \frac{T_1}{T_E}$ |
| **Efficiency (E)** | $E = {}^{S}\!/_{p}$ |
| **Score ($S_E$)** | $S_E = \frac{1}{x}\sum_{1}^{x} E_x$ |

## 5. EXPERIMENTAL EVALUATION

In order to evaluate PS workflows using different execution models, we implemented four distinct models to execute the experimental study on top of Chiron workflow engine. The execution models use different task dispatching and workflow orchestration strategy. Each different execution model resembles a different HPC approach. Thus, the main purpose of this study is to evaluate the performance differences between these four execution models in a high performance computing cluster with shared disk storage.

To achieve our main goal, we performed four analyses. The first used small scaling factors to check if the results would show differences in performance even using small input parameter spaces and short-term activities. The second analysis increases ACF to see how it impacts the results. The third analysis uses the *Map* PS workflow pattern to evaluate the scalability of the four execution models and the fourth analysis evaluates the efficiency of the models for four different scaling factors configuration fixing the number of cores. The computational environment used during the experiments is a SGI Altix ICE 8200 distributed memory shared-disk cluster with 32 nodes, each one with 2 quad-core processors and 8 GB of memory. The cluster runs GNU/Linux 2.6.32.12-0.7 x86_64.

The four implemented execution models are the combination of two variable characteristics: task dispatching and workflow data transfer orchestration strategy. The task dispatching strategy can be static (STA) or dynamic (DYN). Static means that the model sends bags of tasks to the working processors, which need to process all the tasks in the bag before asking for new tasks. The dynamic strategy sends one task at a time as the working processors becomes idle. Static and dynamic strategies had already been studied before [27–29]. The second characteristic, the workflow data transfer orchestration strategy, can vary from *First Activity First* (FAF) to *First Tuple First* (FTF) [21]. FAF means that all the task of the first activity needs to be processed (i.e. all its data generated) before the start of the second activity, and so on. FTF, though, lets the PS workflow run in parallel on a pipeline fashion, running the tasks as any part of their input data becomes available. The FAF characteristic can be related to Intra-Operator Parallelism in parallel database systems while FTF can be related to Inter-Operator Parallelism [13]. The combination of Static, Dynamic, FAF and FTF characteristics creates the four execution models as seen in Table 5.

**Table 5. Execution models**

| | Static | Dynamic |
|---|---|---|
| First Activity First | STA_FAF | DYN_FAF |
| First Tuple First | STA_FTF | DYN_FTF |

The STA_FAF resembles a MapReduce approach combined with a general SWfMS (e.g., VisTrails). DYN_FAF model is similar to a traditional SWfMS combined with a specialized HPC middleware (e.g., Nimrod/K [30]). STA_FTF is similar to pipeline DAG workflow schedulers, such as Pegasus [31], while DYN_FTF is more similar to parallel SWfMS, such as Swift [8]. Although we are not using different tools for each different execution model, our objective is to evaluate the behavior of PS workflows execution raising differences between those execution models. These differences evidence the need of a workflow engine that supports different execution models and an optimizer that can choose the model according to the PS workflow scaling factors.

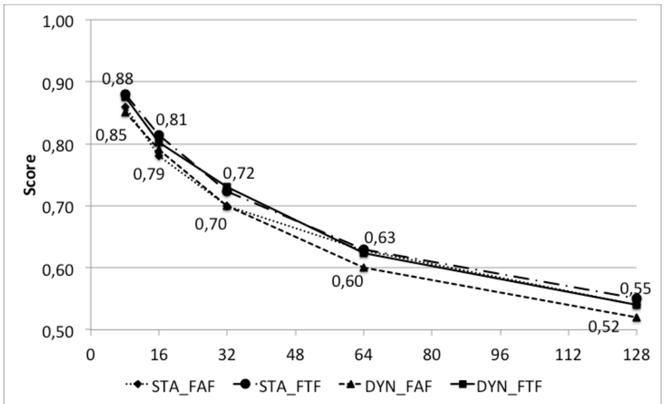


**Figure 12. Evaluation results using ACF=1 and ISF=1**

In our first analysis, we considered the smallest possible scaling factor for the PS workflow patterns, considering both ACF=1 and ISF=1. These factors generated 512 input parameters for each PS workflow pattern, with activities that take an average of 2 seconds to be processed. The purpose of this analysis is to evaluate if it is possible to raise performance differences even for small datasets and short-term activities. We measured the execution time for all PS workflow patterns using 8, 16, 32, 64 and 128 cores. Then, we computed the score for each execution model as the average of the efficiency of each PS workflow pattern execution. The results are shown on Figure **12**. To make the charts clearer we are showing data labels for the execution models with best and the worst performance. In this case, we depict the labels for STA_FTF that got the best performance and for DYN_FAF that got the worst.

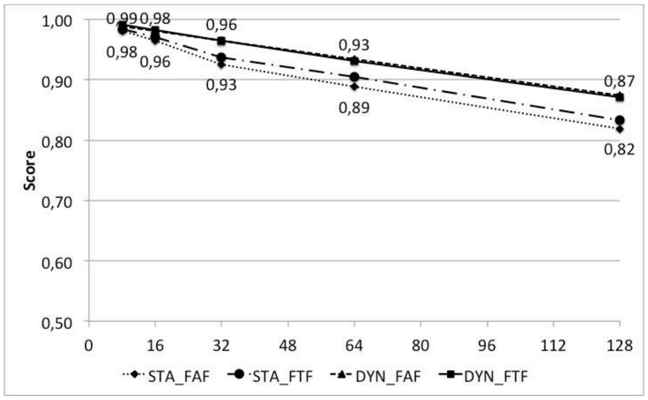


**Figure 13. Evaluation results for ACF=6 and ISF=1**

The decrease in the score as we increase the number of cores is expected due to the small quantity of parallel tasks per PS workflow pattern and the short-term activities. The two static execution models performed slightly better than the dynamic ones, when it comes to the average efficiency of the parallelism (score). However, differences in total execution time between the two approaches in some specific cases are more than twenty five percent. This is interesting to observe because, since we have short-term tasks, it is more efficient to dispatch bags of tasks often than to dispatch one task at a time repeatedly. The static models reduce the communication overhead to transmit task over the working nodes. The FTF model also performed better than FAF. This is reasonable because it reduces the time the working nodes wait to obtain new tasks, since they do not need to wait until a given activity finished to obtain tasks from the next activity.

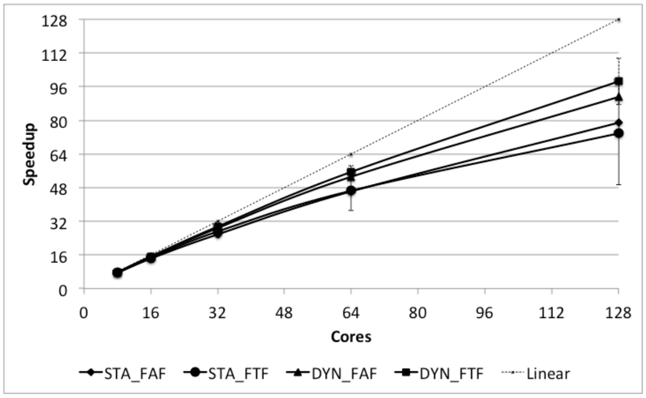


**Figure 14. Average speedup for the Map PS workflow pattern**

In the second analysis, we used ACF=6 and ISF=1. This scenario evaluates the impact of the activity cost. Using ACF=6, the average duration of an activity would be around 64 seconds. The results are shown on Figure 13. Again, we used the score metric as the average of the efficiency of each PS workflow pattern execution. Data labels are shown for the DYN_FTF and STA_FAF execution models. The overall score decreases and we see that dynamic models are more efficient than static ones. This is expected since dynamic models are more flexible and reduce the idleness of working nodes, sending new tasks as they need, providing a working load balance.

The first two analyses showed how we could present performance differences between the execution models. Also, varying the scale factor produces differences in the results, as desired. It is possible to manipulate the scaling factors ACF and ISF to evaluate an execution model according to the characteristics of an experiment. Additionally, the results may reflect the behavior of the model under a given computational environment.

The third analysis aims at evaluating the scalability of the execution models. We have used a single PS workflow pattern, *i.e.* a workflow that is a combination of *Map* patterns. This is interesting because it checks if a single PS workflow pattern is capable of revealing the most suitable execution model to use on a specific workflow pattern. For this third analysis, we used four combinations of ACF and ISF as presented in Table 6. The results present the average speedup for each execution model and are shown on Figure 14. To maintain the clarity of the graph, we are showing error bars only for the worst and best results. All execution models scale well but the speed degrades specially after 64 cores. This may be due to the small size of the input parameter space, *i.e.*, there are too many cores to process and not so many input data. This scenario reduces efficiency since it increases idleness in the working nodes. This is reinforced by the fact that the dynamic models performed better, since their dispatching strategy aims at reducing the overall idleness of the execution. The DYN_FTF is especially good in this scenario since there is almost no waits to dispatch a new task.

**Table 6. ACF and ISF values for analysis three and four**

| | | | | |
|---|---|---|---|---|
| **ACF** | 2 | 2 | 4 | 6 |
| **ISF** | 1 | 2 | 1 | 1 |

The last analysis uses the same configuration of ACF and ISF described in Table 6. However the number of cores is fixed to 32 since the previous analysis showed it is the configuration that best suits the chosen scaling factors. Besides, this analysis aims at evaluating the score of each parallel execution model for each individual combination of ACF and ISF, so we can have a deeper look at their impact on the execution models. The results are shown on Figure 15. The dynamic models are the more efficient parallel execution models for all given scenarios since it is the most flexible model, naturally reducing the idleness in the working nodes. Differences in score between the evaluated execution models are up to 14%. Also, we see that greater ACF or ISF are the targets of parallel computing thus more efficient. This is reasonable since the cost benefit of dynamic distribution and communication overhead does not pay for small quantity of short-term tasks.

Our evaluation was capable of performing different types of analysis regarding what parallel execution model best suits different experiment characteristics. It offers an overview of the behavior of the models using a given ACF and ISF as we performed in our first and second analysis. It was also possible to evaluate the speedup of the execution models for a specific PS workflow pattern of greater interest, like we did in the third analysis. It is also possible to make a deeper investigation varying ACF and ISF for all PS workflow patterns to check their impact on the score of the execution models on a set of cores of the computing environment.

With this study we evaluated the behavior of PS workflow using different execution models in an HPC cluster with shared disk storage. The study shows that, in general, the DYN_FTF execution model is the most efficient parallel execution model in most situations. However, if the PS workflow deals with short-term activities, it may be interesting to use a static model to reduce task-dispatching overhead. FTF models also tend to be more efficient than FAF in most situations although sometimes they present very close performance results. Besides, we noticed considerable elapsed time differences between the models when running different combinations of ACF and PS workflow patterns. Thus it may be more interesting to have all execution models available in the workflow engine. Scientists and workflow designers can use this study as a preliminary analysis to choose the best execution model to run their specific workflows in parallel. Indeed, our evaluation study design is a step towards a benchmark for PS workflows. More information about this project can be found in the Scientific Workflow Benchmark home page [32].

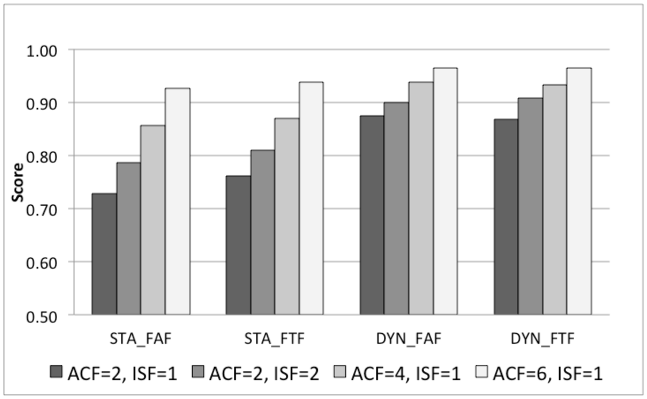


**Figure 15. Evaluation results using four combinations of ACF and ISF but fixing the number of cores**

# 6. RELATED WORK

Although there has been much work on workflows and workloads characterization in general, to the best of our knowledge, there has been very little work focused on performance comparison on data-intensive workflows such as PS workflow. The work by Bharathi *et al.* [20] describes basic workflow structures that are composed into complex scientific workflows, providing a characterization of workflows and presenting several real scientific applications. However, they do not present an evaluation using their workflow patterns on different execution models in order to measure their performance behavior. Other studies [33,34] evaluate workloads to characterize different types of workflows in grid infrastructures but also do not present different comparison metrics nor evaluations using different execution models. The work by Gillman *et al.* [35] proposes a benchmark that specifies an e-commerce workflow, in order to stress the components of a Workflow Management System. Despite the fact that our study and Gillman's benchmark share some principles of measuring the performance of execution model approaches, their work is related to business workflows, *i.e.* more control-centric, considering structures such as conditional branching. Our focus is on PS workflows, *i.e.* data-centric workflows, which may involve long-term activities.

The work of Vassiliadis *et al.* [36] propose a unified way to construct and measure the efficiency and the effectiveness of Extraction-Transform-Load (ETL) workflows. They are centered in ETL processes, proposing a principled organization of workflow patterns for this scenario. Goderis *et al.* [37] introduce a study in the field of scientific workflow discovery, to identify the practice and attitudes of scientists. From this study, they establish three benchmarks, focused on the requirements of semantic workflow discovery tools. They do not stress different execution models to evaluate the performance of workflow execution though. Another interesting initiative is the Yahoo! Cloud Serving Benchmark (YCSB) [38]. It aims at facilitating performance comparisons between cloud data serving systems and it is restricted to cloud computing environments. Besides, it is not related to PS workflows; it aims at a more generic performance evaluation for data transactions over the cloud.

There are also several tools for HPC that can be used for performance comparisons. The LINPACK benchmark [39] aims at measuring the performance of an HPC environment by solving various systems of linear equations, and the results are mainly reported through the amount of floating point operations per second. A portable implementation of this benchmark, HPL, is used in the TOP500 project, which ranks and details supercomputers. The NAS Parallel Benchmarks (NPB) [40] comprises a set of benchmarks for the performance evaluation of highly parallel supercomputers. These benchmarks consist of five parallel kernels and three pseudo-applications, in order to simulate the computation and the data movement characteristics of computational fluid dynamics applications. Both HPL and NAS are designed to stress HPC environments to measure their performance. However, their applications are not suitable for PS workflows since they are not designed for parameter exploration experiments. They also do not show different parallel workflow execution models. We can also find in the literature some work with regard to MapReduce systems, such as MRBench [41] and HiBench [42]. It is important to notice, however, that these initiatives intend to measure the performance of HPC architectures, and not the performance of execution models to execute PS workflow in HPC, which is our focus. Additionally, we observed that PS workflows might have different performance in the same architecture, which reinforces the need for PS evaluations on different parallel execution models.

## 7. CONCLUSION

In this paper, we studied the behavior of typical PS workflows to provide a reference for workflow engine designers or optimizers in identifying the most suitable execution model. We defined five PS workflow patterns and four types of activities based on workflow typical characterizations. Each PS workflow pattern corresponds to a specific workflow pattern and can be used to evaluate the performance of different parallel execution models under different distributed computing environments such as clusters, grids or clouds. This way, the results reveal the probable behavior of PS workflows when using the evaluated execution model, helping to identify the most adequate model for different scenarios.

To validate our proposal, we used the designed PS workflow patterns to evaluate and compare four execution models on a shared-disk HPC cluster. The main goal of our study is to identify variations in performance of these four different execution models. Our first two analyses detected performance differences for small datasets with both short-term and long-term activities. Our third analysis measured the scalability of the execution models on a specific PS workflow pattern and the last analysis had a deeper look on the behavior of the execution models for different combinations of the scaling factors ACF and ISF.

Overall, the results show that dynamic execution models are more suitable for most PS workflow executions. However, static execution models are more efficient when dealing with short-term activities. Besides, static execution models tend to provide better performance, due to its simplicity. Considering that most PS workflows need HPC, the proposed study can be used as a basis for selecting the best execution model for a given PS workflow. Additionally, our study is a step towards a benchmark specially designed for the evaluation of PS workflow execution performance, particularly in HPC.

## 8. ACKNOWLEDGMENTS

We would like to thank the High Performance Computing Center (NACAD-COPPE/UFRJ), where the experiments were executed.

## 9. REFERENCES

[1] W. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems*. The MIT Press, 2002.

[2] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[3] M. Mattoso, C. Werner, G.H. Travassos, V. Braganholo, L. Murta, E. Ogasawara, D. Oliveira, S.M.S. da Cruz, and W. Martinho. Towards Supporting the Life Cycle of Large-scale Scientific Experiments. *International Journal of Business Process Integration and Management*, 5(1):79–92, 2010.

[4] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M.R. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34(2):729–732, 2006.

[5] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: an extensible system for design and execution of scientific workflows. *Scientific and Statistical Database Management*, 423–424, 2004.

[6] S.P. Callahan, J. Freire, E. Santos, C.E. Scheidegger, C.T. Silva, and H.T. Vo. VisTrails: visualization meets data management. *SIGMOD International Conference on Management of Data*, 745–747, 2006.

[7] Y. Gil, V. Ratnakar, E. Deelman, G. Mehta, and J. Kim. Wings for Pegasus: Creating Large-Scale Scientific Applications Using Semantic Representations of Computational Workflows. *The National Conference On Artificial Intelligence*, 1767–1774, 2007.

[8] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde. Swift: Fast, Reliable, Loosely Coupled Parallel Computation. *3rd IEEE World Congress on Services*, 206, 199, 2007.

[9] E. Walker and C. Guiang. Challenges in executing large parameter sweep studies across widely distributed computing environments. *Workshop on Challenges of large applications in distributed environments*, 11–18, 2007.

[10] E. Ogasawara, P. Valduriez, and M. Mattoso. *Chiron: Scientific Workflow Engine*, http://datluge.nacad.ufrj.br/chiron, 2011.

[11] J. Dean and S. Ghemawat. MapReduce: a flexible data processing tool. *Commun. ACM*, 53:72–77, 2010.

[12] E. Ogasawara, D. Oliveira, F. Chirigati, C.E. Barbosa, R. Elias, V. Braganholo, A. Coutinho, and M. Mattoso. Exploring many task computing in scientific workflows. *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, 1–10, 2009.

[13] M.T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. 3 ed. New York, Springer, 2011.

[14] N. Russell, A.H.. Ter Hofstede, W.M.. van der Aalst, and N. Mulyar. Workflow control-flow patterns: A revised view.

*BPM Center Report BPM-06-22, BPMcenter. org*:06–22, 2006.

[15] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.

[16] L.M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.

[17] W. Gropp, E.L. Lusk, and A. Skjellum. *Using MPI - 2nd Edition: Portable Parallel Programming with the Message Passing Interface*. second edition ed. The MIT Press, 1999.

[18] I. Raicu, I.T. Foster, and Yong Zhao. Many-task computing for grids and supercomputers. *Proceedings of the Workshop on Many-Task Computing on Grids and Supercomputers*, 1–11, 2008.

[19] L. Bouganim, D. Florescu, and P. Valduriez. Dynamic load balancing in hierarchical parallel database systems. *Proceedings of the 22nd International Conference on Very Large Databases*, 436–447, 1996.

[20] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi. Characterization of scientific workflows. *Workflows in Support of Large-Scale Science*, 1 –10, 2008.

[21] E. Ogasawara, J. Dias, D. Oliveira, F. Porto, P. Valduriez, and M. Mattoso. An Algebraic Approach for Data-Centric Scientific Workflows. *Proc. of VLDB Endowment*, 4(12):1328–1339, 2011.

[22] ProvChallenge. *Provenance Challenge Wiki*, http://twiki.ipaw.info/bin/view/Challenge/WebHome, 2010.

[23] F. Coutinho, E. Ogasawara, D. Oliveira, V. Braganholo, A.A.B. Lima, A.M.R. Dávila, and M. Mattoso. Many task computing for orthologous genes identification in protozoan genomes using Hydra. *Concurrency and Computation: Practice and Experience*, 23(17):2326–2337, 2011.

[24] H. González-Vélez and M. Leyton. A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers. *Softw. Pract. Exper.*, 40(12):1135–1160, 2010.

[25] W. van der Aalst, A. Hofstede, B. Kiepuszewski, and A. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[26] S. Callaghan, P. Maechling, P. Small, K. Milner, G. Juve, T.H. Jordan, E. Deelman, G. Mehta, K. Vahi, et al. Metrics for heterogeneous scientific workflows: A case study of an earthquake science application. *International Journal of High Performance Computing Applications*, 25(3):274 –285, 2011.

[27] J. Yu and R. Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing*, 3(3-4):171–200, 2006.

[28] R. Prodan and T. Fahringer. Dynamic scheduling of scientific workflow applications on the grid: a case study. *Proceedings of the 2005 ACM symposium on Applied computing*, 687–694, 2005.

[29] C. Pautasso and G. Alonso. Parallel computing patterns for Grid workflows. *Workflows in Support of Large-Scale Science, 2006. WORKS '06. Workshop on*, 1 –10, 2006.

[30] D. Abramson, C. Enticott, and I. Altinas. Nimrod/K: towards massively parallel dynamic grid workflows. *Proc. of International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–11, 2008.

[31] E. Deelman, G. Mehta, G. Singh, M.-H. Su, and K. Vahi. Pegasus: Mapping Large-Scale Workflows to Distributed Resources. *Workflows for e-Science*, , Springer, 376–394, 2007.

[32] SWB. *SWB - Homepage*, http://datluge.nacad.ufrj.br/swb, 2011.

[33] D. Thain, J. Bent, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, and M. Livny. Pipeline and batch sharing in grid workloads. *12th IEEE International Symposium on High Performance Distributed Computing, 2003. Proceedings*, 152– 161, 2003.

[34] S. Ostermann, R. Prodan, T. Fahringer, R. Iosup, and D. Epema. On the Characteristics of Grid Workflows. *Proceedings of the CoreGRID Workshop on Integrated Research in Grid Computing (CGIW'08)*:431–442, 2008.

[35] M. Gillmann, R. Mindermann, and G. Weikum. Benchmarking and Configuration of Workflow Management Systems. *In Cooperative Information Systems, 7th International Conference, COOPIS 2000, Eilat, Israel*:186–197, 2000.

[36] P. Vassiliadis, A. Karagiannis, V. Tziovara, A. Simitsis, and I. Hellas. Towards a Benchmark for ETL Workflows, 2007.

[37] A. Goderis, U. Sattler, P. Lord, and C. Goble. Seven Bottlenecks to Workflow Reuse and Repurposing. *The Semantic Web – ISWC 2005*, 323–337, 2005.

[38] B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems with YCSB. *Proceedings of the 1st ACM symposium on Cloud computing*, 143–154, 2010.

[39] A. Petitet, R. Whaley, J. Dongarra, and A. Cleary. *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers* Available at: http://www.netlib.org/benchmark/hpl/, 2010.

[40] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, et al. The Nas Parallel Benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63 –73, 1991.

[41] Kiyoung Kim, Kyungho Jeon, Hyuck Han, Shin-gyu Kim, Hyungsoo Jung, and H.Y. Yeom. MRBench: A Benchmark for MapReduce Framework. *14th IEEE International Conference on Parallel and Distributed Systems, 2008. ICPADS '08*, 11–18, 2008.

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