A Grid Simulation Framework to Study Advance Scheduling Strategies for Complex Workflow Applications

Adan Hirales-Carbajal, Andrei Tchernykh
Computer Science Department
CICESE Research Center
Ensenada, Baja California, Mexico
e-mail: ahirales@uabc.mx, chernykh@cicese.mx

Thomas Röblitz, Ramin Yahyapour
IT und Medien Centrum & Fakultät für Informatik
Technische Universität Dortmund
Dortmund, Germany
e-mail: thomas.roeblitz@udo.edu,
ramin.yahyapour@.udo.edu

***Abstract*—Workflow scheduling in Grids becomes an important area as it allows users to process large scale problems in an atomic way. However, validating the performance of workflow scheduling strategies in real production environment cannot be feasibly carried out. The complexity of production systems, dynamicity of Grid execution environments, and the difficulty to reproduce experiments, make workflow scheduling production systems a complex research environment. Instead, this work is based on a trace driven simulator.**

This work presents workflow scheduling support as an extension to the Teikoku Grid Scheduling Framework (tGSF). tGSF was developed as a response for a standard compliant and trace based Grid scheduling simulation environment. Workflow scheduling is provided via a second layer of Grid scheduler, extensible to new workflow and parallel scheduling strategies. This work also includes a usage case scenario, which illustrates how this extension can be used for quantitative experimental study.

Keywords: Grid computing, workflow scheduling, scheduling

I. INTRODUCTION *(HEADING 1)*

A workflow developed to describe business processes and problems in commerce, science, and engineering is the "automation of a process, which involves the orchestration of a set of grid services, agents and actors that must be combined together to solve a problem or to define a new service" [1]. Workflows are predominantly characterized by aggregating data and functionality into independent atomic-like jobs.

Perhaps, the most primitive example of a workflow is a batch file. When submitted each job in the batch is executed one after the other, therefore precedence constraints are chain like. Contemporary workflows are conceptually modeled as Graphs or DAGs, when no cycles are present in the workflow. In production systems, workflows are generally modeled by means of a job description language for portability of descriptions. Some examples include XPDL, BPEL, and DAX [2-4].

Workflows are submitted as atomic jobs, failure of a job within the workflow compromises the execution. Nevertheless, many exception handling mechanisms have been proposed to deal with this problem [24-26]. In this work, no exception handling is supported. In the event an exception occurs, the workflow execution is terminated.

Workflow scheduling is a process in which resources are allocated to workflows jobs. Common resources whose usage can be virtualized include processors, disk space, and network bandwidth. The workflow scheduler multiplexes virtual resources (virtual machines) so that concurrent access to shared resources is attained.

Production systems with workflow scheduling support include Pegasus, DAGMan/Condor, and Karajan/Globus [5-7]. However, validating the performance of workflow scheduling strategies in such infrastructures is complex and time consuming. To perform such a task, researchers often require access to resources, source code, and instrumentation mechanism. Furthermore, workload and resource utilization conditions must be reproducible, because validation is essentially a parameter-sweep application.

Although many workflow scheduling production systems exist to enable computational and storage capabilities in a transparent and scalable way, it is difficult to use them for experimental research. Thus simulation is a feasible method for studying workflow scheduling problems.

In this paper, we present an architecture and design features of a workflow and parallel job scheduling simulation framework developed as an extension to Teikoku Grid Scheduling Framework (tGSF). tGSF is a generic standard based parallel scheduling framework. It provides mechanisms for parallel job interchange between sites in a computational Grid. Site acceptance and distribution of jobs are subject to policies; queuing and scheduling strategies are configurable; provenance information is associated to jobs during their execution life cycle and optionally stored to permanent storage. It is based on the Standard Workload Format (SWF). tGSF is a flexible configurable and extensible parallel Grid scheduling framework that complies with standards and offers a controllable research environment. Teikoku version 0.1 is available from [10].

In this work, tGSF has been extended with the capabilities of: (1) scheduling workflows and parallel jobs at the Grid layer; (2) query real-time or stale site state information; (3) query job execution time estimates; and (4) monitoring/storing performance accounting data. With such functionality, tGSF users can perform performance evaluation studies that include scheduling of parallel and

workflow jobs. The architectural design for supporting the previously mentioned functionality, is the contribution of this work.

The paper is organized as follows. Section II presents general architecture features of tGSF. Section III gives notation and formally describes the scheduling problem. Section IV describes the added architectural features that support workflow and parallel job scheduling at the Grid layer. Section V illustrates the simulation environment. Section VI discusses related work. Conclusion and future work are presented in Section VII.

II. THE TGSF GRID SCHEDULING FRAMEWORK

The tGSF was developed to provision a simulation framework for Grid scheduling. It is a Java based application developed by the Grid Scheduling Architecture Research Group (GSA-RG) of the open Grid Forum and a research group of equal name within the CoreGrid [8]. tGSF uses, wherever possible, standards for workload representation, performance metrics, and the scheduling architecture [9]. It was initially designed for parallel job interchange between computational sites. It uses job acceptance, distribution, and location policies to achieve load balancing. tGSF is structured in the following four layers (see Fig.1):

- *Foundation layer* is an event-driven kernel that manages global time and event dispatching. Timed events are registered and dispatched by the kernel. The run time environment allows carrying out real-time, simulation, or debugging scheduling setups.
- *Commons layer* provides an abstraction for modeling of jobs, metrics, and persistence. A job is an aggregation of a description, life cycle, and provenance information. The description holds static attributes that define job ownership, group membership, and resource requirements. The job-cycle holds information that describes job current and historic states. Provenance stores the job execution path from the job released site to the location of the host that satisfied the resource request. Metrics provide mechanisms for evaluating job performance. Lastly, persistence provides mechanisms to access permanent storage, such as, relational data bases and files.
- *Site layer*. Resource administration is performed at site layer by means of an abstract scheduler. Strategies are used to advice the scheduler of possible job assignments. The scheduler can evaluate multiple strategies and select the most appropriate one. Parallel job scheduling strategies such as easy backfilling and FCFS are used. This layer also provides data warehousing to facilitate retrieving resource state information, used by strategies to make job allocation decisions.
- *Grid layer*. Previous version of tGSF does not provide Grid scheduling support. It enables job interchange between sites by means of acceptance, location, and distribution policies. The decision maker may be operated under different settings. In a centralized set-up, its responsibility is to accept jobs and delegate them to the local site scheduler. In a decentralized set-up, job delegation is also performed based on a distribution policy.

A site functioning or role may be oriented on computational, data, or memory intensive tasks. Only computational sites are considered here. Computational sites provide processing capabilities to execute parallel jobs. A computational site is an ensemble of components from the commons, site, and Grid layer, for instance: an activity broker, parallel scheduler, job submission component, metrics, an information provider, and acceptance and distribution policies.

Parallel jobs are submitted to computational sites via a local submission component or forwarded from an external site. Locally submitted jobs may be processed or delegated to other sites. The delegated job acceptance or denial is determined by local acceptance policies.

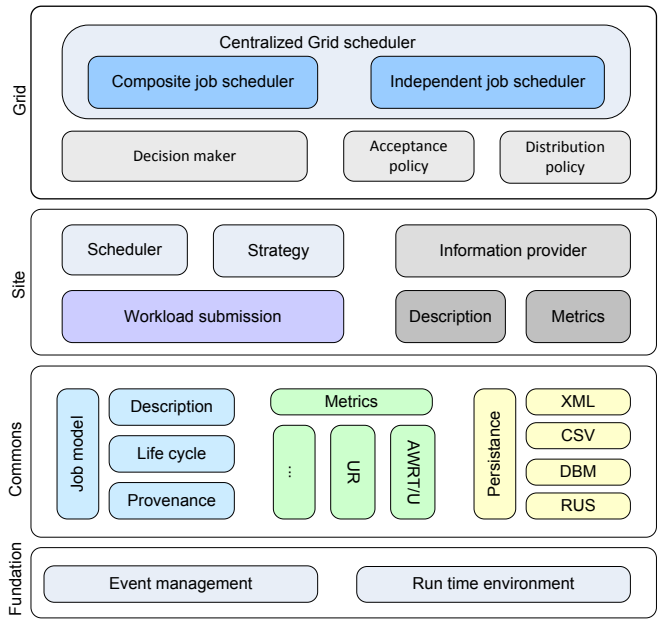

Figure 1. tGSF architecture. Workflow scheduling support is provided at the Grid layer by means of the composite and independent job schedulers.

In this work, we extend tGSF scheduling capabilities by adding workflow and parallel job scheduling support at the Grid layer. A *centralized grid scheduler* is responsible for orchestrating job placement to computational sites in the Grid. Two levels of scheduling are distinguished, namely, site scheduling and grid scheduling (see Fig. 2). The Grid scheduler queues parallel jobs and jobs with broken precedence constraints. It also buffers workflow control and state information used during workflow execution.

III. THE SCHEDULING PROBLEM

We address an online scheduling problem, in which n jobs $J_1, J_2, \cdots, J_n$ must be scheduled to m parallel machines $N_1, N_2, \cdots, N_m$ referred to as computational sites. m_i denotes

the number of identical processors of machine N_i. Machines are ordered in ascending order of their sizes, so that $m_1 \leq m_1 \leq \cdots \leq m_m$.

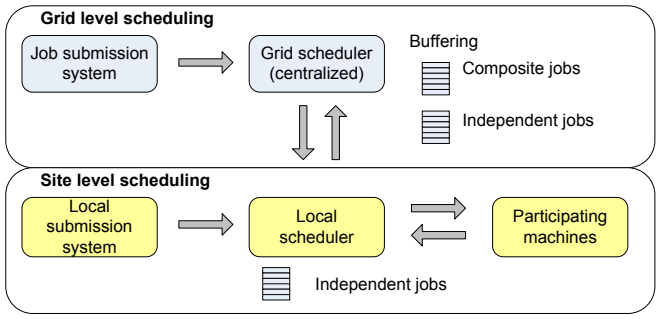

Figure 2. tGSF new two layer scheduling architecture.

Job J_j can be a parallel rigid or composite workflow job. Each parallel job J_j is described by the 4-tuple $(r_j, size_j, p_j, \bar{p}_j)$, with release time $r_j \geq 0$, size or degree of parallelism $1 \leq size_j \leq m_m$, execution time p_j, and user runtime estimate $\bar{p}_j$.

Each composite job J_j is represented by a directed acyclic graph $DAG_j = (V_j, E_j)$, were $V_j = \{J_k, k = 1 \cdots v\}$ is the set of rigid jobs with $|V_j| = v$, $E_j = \{(J_k, J_l) | J_k, J_l \in V_j\}$ is the set of precedence constraints. It is described by the 4-tuple $(r_j, DAG_j, p_j, \bar{p}_j)$ its release time $r_j \geq 0$, $r_j = min\{J_{entry}\}$, $J_{entry} \in V_j$ jobs with no predecessors, its execution time $p_j = \sum_{k=1}^{v} p_k$, and estimated processing time $\bar{p}_j = \sum_{k=1}^{v} \bar{p}_k$. Jobs in V_j are scheduled to different sites, co-allocation of a rigid jobs onto separate sites is not allowed. Hereafter, rigid and composite jobs are referred to as jobs.

IV. Grid Scheduling Support

A. General Aspects of Workflow Scheduling

Site level schedulers in Teikoku receive jobs that arrive over time, and schedule them without knowledge of the future, so that there are no guarantees of producing optimal schedules. Therefore, tGSF falls under the online scheduling model.

Based on the fact that numerous deterministic workflow scheduling strategies bare common functionality, we have procured the design of a loosely coupled workflow scheduling architecture. Static workflow –DAG- scheduling strategies generally employ three phases [11]:

- **Labeling phase** prioritizes each job in the workflow. Priorities are later used to establish the order in which jobs are to be scheduled. Classical labeling strategies include: upper rank, downward rank, and as late as possible.
- **Job selection phase.** Jobs are selected based on the objective of the selection criteria, higher or lower priority jobs are selected first and passed to the assignment phase.
- **Job assignment phase.** An optimization function is used to determine the best location for job placement. Typical optimization functions include: earliest start time and absolute latest start time.

These three phases have been rigorously used in many designs of static and dynamic DAG scheduling strategies [11-14]. In this work, prioritization, selection, and assignment phases are defined in an interface. Furthermore, phases are implemented as policies. For instance, prioritization phase policies can be downward rank, upper rank, as late as possible, or other. Such design, enables exploring different scheduling settings. To this writing, we have extended tGSF to support two workflow scheduling strategies, namely HEFT (Heterogeneous Earliest Finishing Time) and CPOP (Critical Path on Processor) [11]. Components that enable workflow support and their interactions are subject of the following section.

B. Components for Workflow Scheduling

The primary goals in the design of the proposed workflow scheduling environment are to provide an extensible, highly cohesive, and loosely coupled infrastructure. Extensibility is supported by using wherever possible interfaces or abstract classes that provides specializing functionality, such as scheduling policies, optimization functions and metrics, schedulers, and job administration brokers. High cohesion and loosely coupling are partially attained by making components functionality self contained, and providing mechanisms for processing shared data. This objective is achieved by abstracting each job's control and state data into independent Job Control Blocks (JCB). Such a design principle is commonly used to model process and threads in modern operating systems [15, 16].

Workflow scheduling support is based on the scheduling phases described in the previous section. It is an aggregation of specialized schedulers, strategies, information providers, and brokers that orchestrate scheduling of workflows. The following components enable workflow scheduling support:

- **Abstract Grid Broker:** The Abstract Grid Broker (AGB) receives jobs from a grid submission component, and forwards them to a parallel job scheduler or to a workflow scheduler. Several Grid broker implementations are possible, for instance, a distributed, hierarchical, or centralized. Differences between Grid broker implementations may lie in the visibility of resources, interconnection topology, among other factors. A Centralized Grid Broker (CGB) with complete view of all computational sites in the Grid is designed.

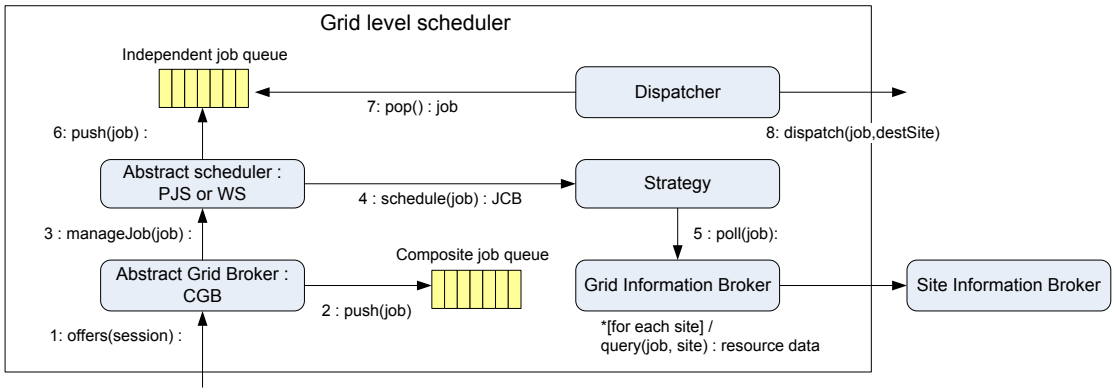

Figure 3. Parallel and composite job tGSF extended Grid scheduling support.

- **Abstract Scheduler**: The Abstract Scheduler (AS) manages jobs by applying a parallel job or workflow scheduling scheme. Two concrete schedulers are distinguished: Parallel Job Scheduler (PJS) and Workflow Scheduler (WS). WS processes workflow jobs by ranking independent jobs, ordering them based on a selection criterion, and determining job destinations. The preceding process is repeated as precedence constraints are broken.
- **Strategy**: A strategy is an extensible interface used to define job assignment policies, for example, random, minimum parallel load, minimum lower bound, etc. Polices are discussed later in section IV.F.
- **Information system:** The Information system (IS) provides a query interface for status information of compute sites and jobs execution time estimates. Assignment policies may be used by some strategies to gather information for decision making purposes. Section IV.E describes the information system design.
- **Dispatcher**. The dispatcher forwards jobs to the destination site determined by a strategy.

The sequence of events performed during the collaboration of workflow scheduling components may not be unique, as workflow scheduling strategies may omit some steps of the scheduling scheme. Others may reevaluate values as the scheduling process progresses. This is the case of dynamic critical path workflow scheduling strategies. In spite of the differences in sequencing of events, we illustrate a general set of events for scheduling workflows in Figure 3 and describe them next.

1. The scheduling process is initiated, when jobs are submitted –offered- to the CGB via a grid submission component. Sessions are used to send one or more jobs as a single request. Session duration is bounded, if session torn down occurs before consuming all session jobs, the CGB does not schedule them. Note that the CGB is a concrete implementation of AGB.
2. Upon reception of a workflow, jobs are *pushed* into the Composite Job Queue (CJQ). The CGB forwards the job to the PJS or the WS based on the job type.
3. The WS manages the job by creating or updating a workflow JCB. Job management includes setting the workflow state, internal workflow jobs states, and storing accounting data. Afterward, a labeling policy is used to rank all workflow jobs. A selection policy criterion is used to select a set of independent jobs ordered according to an ordering policy.
4. Jobs in the independent job set are processed based on their relative ordering. The WS schedules each job by calling an Assignment Strategy (AS) to determine which site will host a given job. Note that one or more assignment strategy can be used.
5. During the execution of the strategy the Grid Information Broker (GIB) polls job execution time estimates (i.e. earliest start time, earliest finishing time, etc.) or site status information (i.e. number of waiting jobs, total running jobs, number of resources, etc.). Ultimately, the strategy selects a destination site and adds localization information to the JCB.
6. As job assignment is determined, the WS pushes jobs into the Independent Job Queue (IJQ), and creates a Job Queued Event (JQE) for the job.
7. On the occurrence of a JQE event, the PJS retrieves –pop- localization information from the IJQ and calls the dispatcher.
8. The dispatcher adds provenance information to the job and dispatches it to its destination site.

In tGSF, the policies and strategies referenced in steps 4 to 6 are defined in an ASCII text based configuration file. They are dynamically loaded during simulation runtime.

C. *Parallel Job Scheduling*

In order to define a suitable scheduling methodology, the following model restrictions are considered: A computational site schedule is only accessible to its local

scheduler. The grid scheduler does not have access to the computational site schedules, it can only request state data or job execution time estimates. Estimates do not guarantee resource availability, nor that reservations performed.

The parallel job scheduling is thus straightforward. Given a parallel or independent job an assignment policy is used to evaluate the candidacy of computational sites. First, a subset of admissible sites is built by selecting computational sites that can fulfill a given job resource requirements. The candidate site is selected according to the optimization criterion of the assignment strategy. Assignment strategies are discussed in Section IV.F. Once the target site is determined, the job is sent to the computational site.

D. The Workflow Model

The original version of tGSF models parallel jobs using the Standard Workload Format (SWF) with chain support [17]. In this work, we extend tGSF abstract job model so that workflows can be scheduled at the Grid layer. We added four new attributes to the existing format, namely: composite job ID, composite job type, vector of successor IDs, and a vector of predecessor IDs. Hereon, we use the term workflow or composite job interchangeably to refer to precedence constraint jobs.

Composite jobs are modeled by Directed Acyclic Graphs (DAG). It is composed by a set of SWF jobs and a set of precedence constrains stored in a hypergraph data type [18]. Such a data type can be used to model jobs with directed, undirected, cyclic, forest, sparse, or other graph properties.

E. Providing Information to Strategies

Scheduling strategies may require *accounting data* or job *execution estimates* for decision-making purposes. In Teikoku, a Grid information component provides services for consulting resource availability, state, and architectural characteristics. Its design is based on the Grid laboratory Uniform Environment (GLUE) schema [19].

Teikoku's Grid information component is composed by a Grid Information Broker (GIB) and the Site Information Broker (SIB). The GIB provides services that enable sampling data from one or more SIB's, whereas SIB retrieves information from local resources.

State and accounting data are stored in a Compute Site Information data structure. tGSF version 0.1 only supports CSI, all do the architecture is extensible so that other resources can be included. State data can accumulate at SIB brokers, therefore becoming stale, or it can be sampled with request. Stale state data is refreshed at user given discrete time frames.

CSI state data hold capacity bounds and accounting information. The first one retrieves quotas such as maximum number of jobs, running jobs, and waiting jobs. Accounting data provide load condition information such as total number of executed jobs, running jobs, waiting jobs, etc.

A job execution estimate gives a prediction assuming a job is placed at a given computational site. SIB provided job execution estimates include earliest start time and earliest finishing time.

The static diagram of the grid information component is illustrated in Fig. 4. Concrete implementations for the site information broker and grid information broker are given by *Impl* suffix named classes.

F. Parallel Job Allocation Strategies

The parallel job Grid scheduling strategies proposed in [20] have been implemented. Tchernykh et al. classified job

TABLE I
RIGID PARALLEL JOB ALLOCATION STRATEGIES

Level	Available information	Allocation criteria	Heuristic
0	The number of machines and their size are known	*Rand*, selects a machine *randomly*	
1	In addition to information in level 0, the number of jobs pending execution at each site is known	*Min_lp*, Selects the site with *minimum processor load*	$min\left(\frac{n_i}{m_i}\right)$
2	In addition to information in level 1, the degree of parallelism of each task is used	*Min_pl*, selects the site with *minimum parallel load.* Where $g(J_k) = M_i$ denotes that job J_k was assigned to machine M_i	$min\left\{\sum_{g(J_k)=M_i} \frac{size_k}{m_i}\right\}$
3	In addition to information in level 2, clairvoyant scheduling is performed by using job estimated execution times.	*Min_lb*, selects the site with *minimum work per processor*	$min\left\{\sum_{g(J_k)=M_i} \frac{size_k \cdot \bar{p}_k}{m_i}\right\}$
4	In addition to information in level 3,all site scheduled are known	*Min_ct*, selects the site with *minimum completion time*. Where $C^i_{max} = max_{g(J_k)=M_i}(C^i_k)$, and C^i_k is the finishing time of job J_k	$min\{C^i_{max}\}$
		Min_wct, selects the site with *minimum weighted completion time*	$min\left\{\sum_{g(J_k)=M_i} size_k \cdot C^i_k\right\}$
		Min_wt, selects the site with *minimum job waiting time*. If n_i is zero n_i is set to 1	$min\left\{\sum_{g(J_k)=M_i} \frac{C^i_k - r_k - \bar{p}_k}{n_i}\right\}$
		Min_wwt, selects the site with *minimum weighted completion time*. The weight is the size of the job	$min\left\{\sum_{g(J_k)=M_i} \frac{(C^i_k - r_k - \bar{p}_k) * size_k}{n_i}\right\}$
		Min_u, selects the site with *minimum overall utilization*. Were $W^i_{total} = \sum_{g(J_k)=M_i} size_k \cdot p_k$ is the total work performed by machine M_i	$min\left\{\frac{W^i_{total}}{C^i_{max} \cdot m_i}\right\}$
		Min st, selects the site with *minimum job start time*	$min\{C^i_j - r_j\}$

allocation strategies based on the amount of information required for decision making. Four levels are distinguished, strategies of level 1, 2, and 3 use site status information, while strategies of level 4 require job execution time estimates. Table I lists parallel job allocation strategies, implemented in the new Grid scheduling framework.

Level 1 throughout level 3 strategies, use the GIB to query site status information. Status information is used by assignment strategies to select a target computational site that will host a given job. Level 4 strategies use the GIB to poll job completion times C_j. Each computational site i that can host a given job j forwards C_j^i to the GIB.

Providing job completion time estimates is expensive, for it may require creating a schedule at each queried site. Furthermore, estimates do not guarantee that computational resources are reserved, nor that the validity of estimates will hold once jobs has been allocated.

G. Performance Metrics

Performance metrics are used to evaluate the performance of the Grid. Performance evaluation is conducted after the termination of an event, normally, after a job completion event or workflow completion event. The performance may also be evaluated during a pre-processing, processing, and post-processing phase. tGSF ver. 0.1 provides support for the evaluation of the average response time, average slowdown, average wait time, average weighted response time, average weighted slowdown, and average weighted wait time at the site layer.

Current parallel job scheduling objectives are summarized in Table II. Parallel job scheduling objectives can be found in classical scheduling theory sources [21,22], workflow scheduling objectives speedup and scheduling length ratio have been used in [11,13]. Workflow scheduling objectives such as waiting time, slowdown, turnaround time, and their weighted and bounded versions are proposed in this work.

Performance metrics are classified as user centric and system centric. Mean waiting time is a system centric metric, the shorter waiting time the faster job execution begins. A parallel job j can be weighted by its processing time p_j, job size $size_j$, or resource consumption $(p_j . size_j)$. Following Schwiegelshohn et al. [23] this ensures that neither splitting nor combination of jobs can influence the performance objective. Hence, job j weight is given by $weight_j = \{p_j, size_j, p_j size_j\}$. A composite job is weighted by its processing time $p_k = \sum_{T_j \in V_k} p_j$, job size $size_k = \sum_{T_j \in V_k} size_j$, or resource consumption $(p_k size_k)$.

Composite job makespan is $|CP_{min}| + AST_k$. The Absolute Start Time (AST_k) of a composite job k is the minimum start time of job k on all computational sites. The critical path $|CP_{min}|$ is the sum of the minimum processing costs of jobs on the critical path.

V. The Simulation Environment

In tGSF version 0.1, the simulation environment is setup by defining a set of sites and their corresponding properties in a Java properties file. Properties are used to define: the number of available resources, session life time, queuing policy, site information broker caching life time, and a set of performance objectives. Properties are set by initiating tGSF system constants, such as: *numberOfProvidedResources*, *informationBroker.class*, etc. System constants can be found in tGSF package API documentation [8].

A site can be configured to function as computational site or as a centralized grid scheduler, this is performed by initiating the *activityBroker.class* or the *gridActivityBroker.class* properties. Only one grid activity broker can be set, since the implementation of the Grid

TABLE II
GRID LAYER PARALLEL JOB AND WORKFLOW PERFORMANCE OBJECTIVES

Level	Metric	Parallel	Workflow
1	Make-span	$C_{max} = max\{c_j \mid T_j \in T\}$	$C_{max} = max\{c_j \mid T_j \in V_k\}$, per workflow
2	Waiting time	$tw_j = c_j - p_j - r_j$	$tw_k = \max_{T_j \in V_k} c_j - \sum_{j=1}^{\lvert V_k \rvert} p_j - \min_{T_j \in V_k} r_j$, per workflow
3	Average waiting time	$\overline{t_w} = \frac{1}{n}\sum_{j=1}^{n} tw_j$	$\overline{t_w} = \frac{1}{w}\sum_{k=1}^{w} tw_k$, where w is the total number of composite jobs
4	Average weighted waiting time	$\overline{t_w} = \frac{\sum_{j=1}^{n} weight_j \cdot tw_j}{\sum_{j=1}^{n} weight_j}$	$\overline{t_w} = \frac{\sum_{k=1}^{w} weight_k \cdot tw_k}{\sum_{k=1}^{w} weight_k}$
5	Slowdown	$SD = \frac{1}{n}\sum_{j=1}^{n} \frac{p_j + tw_j}{p_j}$	$SD = \frac{1}{w}\sum_{k=1}^{w} SD_k$ were $SD_k = \frac{1}{\lvert V_k \rvert}\sum_{T_j \in V_k} \frac{p_j + tw_j}{p_j}$
6	Bounded slowdown	$SD_b = \frac{1}{n}\sum_{j=1}^{n} \frac{max\{10, p_j\} + tw_j}{max\{10, p_j\}}$	$SD_b = \frac{1}{w}\sum_{k=1}^{w} SD_k$ were $SD_k = \frac{1}{\lvert V_k \rvert}\sum_{j=1}^{\lvert V_k \rvert} \frac{max\{10, p_j\} + tw_j}{max\{10, p_j\}}$
7	Turnaround time	$TA = \frac{1}{n}\sum_{j=1}^{n} (c_j - r_j)$	$tr_k = max\{c_j\} - min\{r_i\} \mid T_j, T_i \in V_k$ $TA = \frac{1}{w}\sum_{k=1}^{w} tr_k$
8	Weighted turnaround time	$TA^{weight} = \frac{1}{n}\sum_{j=1}^{n} (c_j - r_j) \cdot weight_j$	$TAw = \frac{1}{w}\sum_{k=1}^{w} (tr_k \cdot weight_k)$
9	Speedup		$SpeedUp_k = \frac{\sum_{T_j \in V_k} p_j}{makespan_k}$
10	Schedule length ration	Selects the site with minimum start time	$SRL_k = \frac{makespan_k}{\sum_{T_j \in V_k} p_j}$

activity broker is centralized.

A single computational site and centralized Grid scheduler site Java properties file configuration excerpt is illustrated in the following pseudo code. Only key configuration features are presented. Documentation of tGSF version 0.1 Java properties can be found in [8].

```
# Site 1 runtime information and properties
# Runtime information
runtime.workloadSource.id = <id1>
runtime.< id1>.associatedSite.ref = site1
runtime.< id1>.url = file:/C:/<workload file>.swf
# Site 1 properties
sites.<id1>.numberOfProvidedResources = 1
sites.< id1>.submissioncomponent.sessionlifetime = 100
# Activity Broker
sites.<id1>.gridActivityBroker.class  = ... CentralizedGridActivityBroker
# Grid information server
sites.<id1>.gridInformationBroker.class =... GridInformationBrokerImpl
# Composite job strategy
sites.< id1>.grid.composite.strategy.class = ... HEFT
# Rigid job strategy
sites.<id1>.grid.rigid.strategy.class =  …Rand
# Optimization function
sites.< id1>.registeredMetric.ref = bounded_slowdown

#Site 2 runtime information and properties
# Runtime information
runtime.workloadSource.id = <id2>
runtime.< id2>.associatedSite.ref = site2
runtime. <id2>.url = file:/C:/<workload file>.swf
# Site2 properties
sites.<id2>.numberOfProvidedResources = 100
sites.<id2>.localsubmissioncomponent.sessionlifetime = 100
# Activity Broker
sites.<id-2>.activitybroker.class = …StandardActivityBroker
# Information broker
sites.<id2>.informationBroker.class = … SiteInformationBrokerImpl
sites.<id2>.informationBroker.refreshRate = -1
…
# Acceptance policy
sites.<id2>.activitybroker.acceptancepolicy.class = …
        StandardAcceptancePolicy
# Scheduler and scheduling policy
sites.<id2>.scheduler.class =…ParallelMachineScheduler
sites.<id2>.scheduler.localstrategy.class = …FCFSStrategy
# Queue policy
sites.<id2>.scheduler.localqueuecomparator.class = …
NaturalOrderComparator
sites.<id2>.scheduler.localqueuecomparator.ordering = ascending
# Metrics
sites.<id2>.registeredMetric.ref = art
```

Two brokers are defined, site 1 role is set as a grid activity broker (*CentralizedGridActivityBroker*) and site 2 role is set as a computational site (*StandardActivityBroker*). Workloads are associated to each site by setting the url property, for example, site 1 workload is *runtime.site1.url = file:/C:/gridWorkload.swf.* The number of processors of a site is defined by the property *numberOfProvidedResources.* Site 2 hosts 100 processors.

Recall that the information system is used to provide strategies state or job execution estimates. Site information broker refresh rate *informationBroker.refreshRate.* A value of -1 disables caching state data, it ensures that sampling is always performed. State data become stale if they are buffered more than a predefine amount of time set by the *informationBroker.refreshRate* to a positive integer value greater than one (milliseconds). A value of 600,000 indicates that information queries within a 10 minute time frame.

The centralized Grid scheduler scheduling policies are set by initiating the *grid.composite.strategy* property to HEFT, for workflow scheduling support, and the *grid.rigid.strategy* property to *Rand*, for parallel job scheduling support.

Computational sites may perform load balancing of locally submitted jobs, by setting the transfer, location, and distribution policies properties. This feature may be used to create dynamicity at site level scheduling layer, or may be disable so that once a job is assigned to a computational site, it cannot be delegated to others sites.

VI. Related Work

OPTORSIM is a Java based simulator developed as part of the European DataGrid project. It aims to optimize the scheduling process for highly intensive computation and data Grids. OPTORSIM scheduling strategies include: *random*; *access control* which optimizes job allocation based on the time needed to access all files required by the job; *queue size*; and *queue access cost* which considers the combine access cost of all jobs in the queue. Replication optimization criterias include: no replication; replication based on the LRU (Least Recently Used) and LFU (Least Frequently Used) algorithms; and an auction based strategy that use the binomial economic model and the Zipf economic model for negotiating file replication. OPTORSIM provides time-based and event-based simulations [27].

Xavantes is a distributed process and workflow execution context based on Java-RMI. It provides a programming model based on BEPL (Business Process Execution Language) used to define process elements and precedence constraints. Two workflow scheduling strategies are supported: *static PHC* (Path Clustering Heuristic) and *dynamic PCH*. Computation and communication estimates are used [13].

VII. Conclusions and Future Work

This work presented tGSF extension for workflow and parallel job Grid scheduling support, provided via a second layer of scheduler with parallel job strategies, workflow strategies, and metric extensibility. Workflow scheduling support is loosely coupled, so that job labeling, selection, and assignment strategies are processed independently. It allows the study of different workflow scheduling settings. To promote the simulator utilization in the scientific community, an example of the simulation setup was presented.

This work is an on process work. The Teikoku research community entails many research branches. Our research branch dealt with the study of workflow and parallel job scheduling strategies in an execution context that experiences resource unavailability. In future work, we plan to develop strategies to evaluate resource availability conditions in the Grid, and use such information to schedule workflows predictably.

ACKNOWLEDGMENT

We thank IRF and ITMC groups in Dortmund, Germany for their hospitality, comments, and support resolving theoretical and technical issues. We gratefully acknowledge the assistance of Alexander Fölling.

This work was supported in part by the DAAD - Deutscher Akademischer Austausch Dienst under Grant Section 414, Code number A0774928 and A0903177, and the Research and Postgraduate Department of the Autonomous University of Baja California (UABC).

REFERENCES

[1] Fox, G. C. and Gannon, D. 2006. Special Issue: Workflow in Grid Systems: Editorials. Concurr. Comput. : Pract. Exper. 18, 10 (Aug. 2006), 1009-1019.

[2] Guelfi, N. and Mammar, A. 2006. A formal framework to generate XPDL specifications from UML activity diagrams. In Proceedings of the 2006 ACM Symposium on Applied Computing (Dijon, France, April 23 - 27, 2006). SAC '06. ACM, New York, NY, 1224-1231.

[3] Ma, R., Wu, Y., Meng, X., Liu, S., and Li Pan 2008. Grid-Enabled Workflow Management System Based On BPEL. Int. J. High Perform. Comput. Appl. 22, 3 (Aug. 2008), 238-249.

[4] Deelman, E., Gannon, D., Shields, M., and Taylor, I. 2009. Workflows and e-Science: An overview of workflow system features and capabilities. Future Gener. Comput. Syst. 25, 5 (May. 2009), 528-540.

[5] Deelman, E., Singh, G., Su, M., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G. B., Good, J., Laity, A., Jacob, J. C., and Katz, D. S. 2005. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Sci. Program. 13, 3 (Jul. 2005), 219-237.

[6] Bharathi, S. and Chervenak, A. 2009. Scheduling data-intensive workflows on storage constrained resources. In Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science (Portland, Oregon, November 16 - 16, 2009). WORKS '09. ACM, New York, NY, 1-10.

[7] Stratan, C., Iosup, A., and Epema, D. H. 2008. A performance study of grid workflow engines. In Proceedings of the 2008 9th IEEE/ACM international Conference on Grid Computing - Volume 00 (September 29 - October 01, 2008). International Conference on Grid Computing. IEEE Computer Society, Washington, DC, 25-32

[8] http://www.coregrid.net/, December 2009.

[9] C. Grimme, J. Lepping, A. Papaspyrou, P. Wieder, R. Yahyapour, A. Oleksiak, O. W¨aldrich, and W. Ziegler. Towards a standards-based Grid Scheduling Architecture. CoreGRID Technical Report TR-0123, Institute on Resource Management and Scheduling, December 2007

[10] Christian Grimme, Joachim Lepping, Alexander Papaspyrou, and Alexander Fölling. http://forge.it.irf.tu-dortmund.de/trac/teikoku/wiki/TeikokuArchitecture#Architecture, December 2009.

[11] Topcuouglu, H., Hariri, S., and Wu, M. 2002. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE Trans. Parallel Distrib. Syst. 13, 3 (Mar. 2002), 260-274

[12] Kwok, Y. and Ahmad, I. 1999. Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. 31, 4 (Dec. 1999), 406-471

[13] Bittencourt, L. F. and Madeira, E. R. 2006. A dynamic approach for scheduling dependent tasks on the Xavantes grid middleware. In Proceedings of the 4th international Workshop on Middleware For Grid Computing (Melbourne, Australia, November 27 - December 01, 2006). MCG '06, vol. 194. ACM, New York, NY, 10.

[14] Kwok, Y. and Ahmad, I. 1996. Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. IEEE Trans. Parallel Distrib. Syst. 7, 5 (May. 1996), 506-521.

[15] Silberschatz, A., Peterson, J. L., and Galvin, P. B. 1991 Operating System Concepts (3rd Ed.). Addison-Wesley Longman Publishing Co., Inc.

[16] Tanenbaum, A. 2007 Modern Operating Systems. 3rd. Prentice Hall Press.

[17] Steve J. Chapin, Walfredo Cirne, Dror G. Feitelson, James Patton Jones, Scott T. Leutenegger, Uwe Schwiegelshohn, Warren Smith, and David Talby, ``Benchmarks and Standards for the Evaluation of Parallel Job Schedulers". In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (Eds.), Springer-Verlag, 1999, Lect. Notes Comput. Sci. vol. 1659, pp. 66-89

[18] Joshua O'Madadhain, Danyel Fisher, Tom Nelson. JUNG, Java Universal Network/Graph Framework, http://jung.sourceforge.net/, December 2009.

[19] Balazs Konya, Laurence Field, Sergio Andreozzi. Glue working group, http://forge.ogf.org/sf/projects/glue-wg, December 2009

[20] A. Tchernykh, U. Schwiegelsohn, R. Yahyapour, N. Kuzjurin "Online Hierarchical Job Scheduling in Grids", CoreGRID Symposium in conjunction with EuroPar 2008 Conference, Las Palmas de Gran Canaria (Spain), August 26th-29th

[21] Michael L. Pinedo. 2008. Scheduling Theory, Algorithms, and Systems (3rd Ed.) Springer. Pp. 13-22.

[22] Joseph Y-T. Leung. 2004. Handbook of Scheduling, Algorithms, Models, and Performance Analysis. Chapmann & Hall/CRC. Pp. 1.1-1.8

[23] Grimme, C., Lepping, J., and Papaspyrou, A. 2008. Discovering performance bounds for grid scheduling by using evolutionary multiobjective optimization. In Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (Atlanta, GA, USA, July 12 - 16, 2008). M. Keijzer, Ed. GECCO '08. ACM, New York, NY, 1491-1498.

[24] Casati, F., Ceri, S., Paraboschi, S., and Pozzi, G. 1999. Specification and implementation of exceptions in workflow management systems. ACM Trans. Database Syst. 24, 3 (Sep. 1999), 405-451

[25] Han, M., Thiery, T., and Song, X. 2006. Managing exceptions in the medical workflow systems. In Proceedings of the 28th international Conference on Software Engineering (Shanghai, China, May 20 - 28, 2006). ICSE '06. ACM, New York, NY, 741-750

[26] Mourão, H. and Antunes, P. 2007. Supporting effective unexpected exceptions handling in workflow management systems. In Proceedings of the 2007 ACM Symposium on Applied Computing (Seoul, Korea, March 11 - 15, 2007). SAC '07. ACM, New York, NY, 1242-1249

[27] Cameron, D. G., Carvajal-Schiaffino, R., Millar, A. P., Nicholson, C., Stockinger, K., and Zini, F. 2003. Evaluating Scheduling and Replica Optimisation Strategies in OptorSim. In Proceedings of the 4th international Workshop on Grid Computing (November 17 - 17, 2003). International Conference on Grid Computing. IEEE Computer Society, Washington, DC, 52