# Overhead Analysis of Grid Workflow Applications

Francesco Nerieri, Radu Prodan, Thomas Fahringer, Hong-Linh Truong

*Institute for Computer Science, University of Innsbruck*
*Technikerstraße 21A, A-6020 Innsbruck, Austria*
{nero,radu,tf,truong}@dps.uibk.ac.at

***Abstract*— In this paper we propose a systematic approach to performance analysis of workflow applications on the Grid. We introduce an ideal model for the workflow execution time and explain the difference to the real measured times based on a hierarchy of performance overheads for Grid computing. We describe how to systematically measure and compute the overheads from individual activities to entire workflow applications. We adjusted well-known parallel processing metrics to the scope of Grid computing, comprising speedup and efficiency. We have implemented and largely automatised our analysis approach in the context of the ASKALON Grid application development and computing environment. We present experimental results that show detailed overhead analysis of two real-world workflow applications executed in a national Grid environment.**

## I. Introduction

Computational Grids aim to provide a ubiquitous infrastructure for seamless, dependable, and pervasive access to geographically distributed high-end computational capabilities that are part of different administrative domains. While there is still no consensus on the programming model, the workflow paradigm has gained increasing interest as an important class of applications appropriate for programming the Grid.

Whereas the Grid provides in theory unlimited amount of computational power, efficient utilisation of Grid resources is an open issue that received little attention from the community. Similar to compiler optimisations, Grid schedulers aim to solve complex optimisation problems by applying heuristics for achieving good mappings of complex workflow applications onto the Grid resources that, e.g. minimise the execution time (and the overheads) or maximise the throughput. While scheduling remains an undecidable NP-complete problem, performance analysis tools can take an important role by giving new insight into overheads that have not been previously considered and that can be further reduced through improved heuristics or by restructuring the workflow.

In this paper, we propose a systematic analysis approach that aims to help the Grid application developer and user understand the nature of performance losses within large-scale executions of workflow applications in heterogeneous Grid environments. We introduce a theoretical reference parameter called *ideal execution time* that aims to provide a realistic bound for the lowest (i.e. "fastest") execution time that could be achieved by a workflow application in a certain Grid environment. We define the difference between the ideal execution time and the actual measured time as *temporal overhead*. We present a novel classification of workflow overheads covering middleware, loss of parallelism, and activity-specific overheads, that help the application or the middleware service developer understand the real reason of performance losses. We further divide these overheads in more detailed subclasses that explain the performance problems at a finer granularity. Finally, we adjust several well-known scalability metrics for Grid computing, including speedup and efficiency. For this paper we have implemented our approach as a post-mortem analysis service integrated in the ASKALON Grid application development and computing environment [1].

Our paper is organised as follows. In the next section we describe the generic workflow and Grid models that we use in our analysis. Section III introduces a hierarchical classification of performance overheads and a systematic methodology for instrumentation, computation, and analysis. Section IV presents experimental results of two real-world workflow applications. We summarise the most relevant related work in Section V and conclude the paper in Section VI.

## II. Model

### A. Workflow Model

In this section we give a formal definition of a workflow application that represents the sufficient foundation for modeling all real-world scientific applications that we are currently using as case studies. This more restrictive model has the advantage of allowing the clear definition of many of the heterogeneous overheads that we will present Section III-B.

We model a well-structured *workflow application* as a directed graph $WF = (Nodes, Edges)$, where $Nodes = \{N_1, \ldots, N_n\}$ is the set of workflow nodes and $Edges = \bigcup_{i=1}^{n-1} (N_i, N_{i+1}) \bigcup \{(N_j, N_k) \mid j > k \geq 1\}$ is the set of *control flow dependencies*. The edges in the latter union set model backward dependencies which implement sequential loops. A node *N* can have any of the following types representing *workflow regions*: (1) *computational activity*, denoted in this paper as *CA*; (2) *data transfer activity* denoted as *DA*; (3) *parallel section*, denoted as $N_{Par} = (N_{p1}, \ldots N_{pm})$; (4) *subworkflow*, denoted as $(Nodes_i, Edges_i)$ and recursively defined according this definition.

If the set of activities in a parallel section have the same type, we call it *parallel loop*. For example, the workflow depicted in Figure 4 (see Section IV) contains two parallel loops: $N_{Par1} = (lapw1\_k1, \ldots, lapw1\_kn)$ and $N_{Par2} = (lapw2\_k1, \ldots, lapw2\_kn)$, and a backward edge $(Converged, lapw0)$.

### B. Workflow Execution Model

We base our Grid workflow analysis on the ASKALON Grid application development and computing environment [1]. In ASKALON, the user specifies workflow applications at a high-level of abstraction using a graphical UML-based modeling tool or an abstract XML-based language. The workflow XML specification is then given to the *Enactment Engine* that requests from a *Scheduler* service to "optimally" map the activities onto the available Grid resources using heuristic-based algorithms. To make a mapping decision, the Scheduler invokes a *Resource Broker* to find the computational resources that match the software and hardware requirements of the workflow, and a Performance Prediction service that returns forecast information about the executions time of each activity based on historical data and regression techniques. The *Performance Analysis* service requests for instrumentation to compute the required metrics and overheads. After all these steps are completed, the Enactment Engine submits the workflow activities on the corresponding Grid sites according to the control flow and data flow dependencies, and automatically collects and stores the raw performance data into a performance database for post-mortem analysis and visualisation. Any of these execution steps may fail, in which case the Enactment Engine retries the previous step and considers the failure as an overhead. For example, whenever sites appear or disappear from the Grid environment, a rescheduling event is triggered and a new mapping obtained from the Scheduler.

In this execution model, only the work performed by the workflow computational activities is considered as useful execution time. The time required for any other task is considered a *temporal overhead*.

### C. Grid Model

We consider the Grid as an aggregation of heterogeneous *Grid sites*. A Grid site consists of a number of computers and storage systems that share same *local* security, network, and resource management policies. Computers within a Grid site may comprise single processors, symmetric multiprocessors, massively parallel processors, and workstation clusters that are utilised as a single computing resource.

## III. Workflow Temporal Overheads

The execution of workflow applications on the Grid is prone to a broad set of overheads that must be understood in order to improve the overall performance. We therefore model the real execution time of a Grid workflow *WF* as the sum of a theoretical *ideal time* $T_{WF}^{ideal}$ and a set of *temporal overheads* $TO_{WF}$ that originate from various sources: $T_{WF} = T_{WF}^{ideal} + TO_{WF}$. Describing the sources of the overheads, classifying them in a detailed hierarchy, and measuring them in a systematic manner is the scope of our analysis effort.

### A. Ideal Workflow Execution Time

In this section we introduce a lower bound for the execution time of workflow applications on the Grid, to which we refer as *ideal execution time*. Our analysis definitions begin from a given workflow schedule decided either statically or at runtime.

The *ideal execution time of a sequential computational activity CA*, denoted as $T_{CA}^{ideal}$, is the the fastest wallclock execution time of the underlying operating system (Unix) process(es) on all processors available on the Grid. The *ideal execution time of a data transfer activity* is zero, since we consider it an overhead: $T_{DA}^{ideal} = 0$. The *ideal execution time* of a parallel section $N_{Par}$ is the fastest ideal execution time of all its activities: $T_{N_{Par}}^{ideal} = \min_{\forall CA \in N_{Par}} \{T_{CA}^{ideal}\}$. The *ideal execution time* of a workflow *WF*, introduced in Section II-A, is the sum of the ideal execution times of all its activities $N \in Nodes$: $T_{WF}^{ideal} = \sum_{\forall N \in Nodes} T_N^{ideal}$.

We define the *total overhead* $TO_{WF}$ of a workflow *WF* as the difference between its measured execution time $T$ and its ideal execution time: $TO_{WF} = T_{WF} - T_{WF}^{ideal}$

### B. Overhead Computation

We propose a new detailed hierarchical classification of performance overheads for Grid workflow applications illustrated in Figure 1. When designing and computing this overhead hierarchy, we have paid special attention to the fact that the overheads need to be *non-overlapping*.

Similar to the workflow ideal execution time, we compute the value of any overhead such as load imbalance, serialisation, data transfer, external load, job replication, and so on (see following paragraphs) for any workflow region by adding the overhead values computed for the underlying computational activities, parallel sections, and sub-workflows (see Section II-A): $TO = \sum_{\forall N \in Nodes} TO_N$.

We show in the following how we compute the most important overheads classified in Figure 1 for individual computational activities and parallel sections, while for sub-workflows they can be computed through recursive summations.

*1) Middleware overheads:* We define the *job management* middleware overhead of a computational activity *CA* as: $TOJM_{CA} = \max_{\forall end(CA)} \{end(CA)\} - \max_{\forall start(CA)} \{start(CA)\} - T_{CA}$, where $T_{CA}$ is the wallclock time of the activity *CA* measured directly on the Grid execution site using the POSIX `time` command. The *max* operator is applied to all event timestamps since an activity may fail at various stages and, therefore, be retried several times until it succeeds.

We divide the middleware overhead into several smaller overhead categories, each sub-category being computed differently. For example, we compute the *job submission*, *waiting*, *queue residency*, respectively *job polling* overheads as (see Figure 2):

$TOJS_{CA} = \max_{\forall submit(CA)} \{submit(CA)\} - \max_{\forall start(CA)} \{start(CA)\};$

$TOW_{CA} = \max_{\forall pend(CA)} \{pend(CA)\} - \max_{\forall submit(CA)} \{submit(CA)\};$

$TOQR_{CA} = \max_{\forall active(CA)} \{active(CA)\} - \max_{\forall pend(CA)} \{pend(CA)\};$

$TOP_{CA} = \max_{\forall end(CA)} \{end(CA)\} - \max_{\forall active(CA)} \{active(CA)\} - T_{CA}.$

We define the overhead of the other ASKALON Grid middleware services *MS*, including the Resource Broker, the Scheduler, and the Performance Prediction service using a

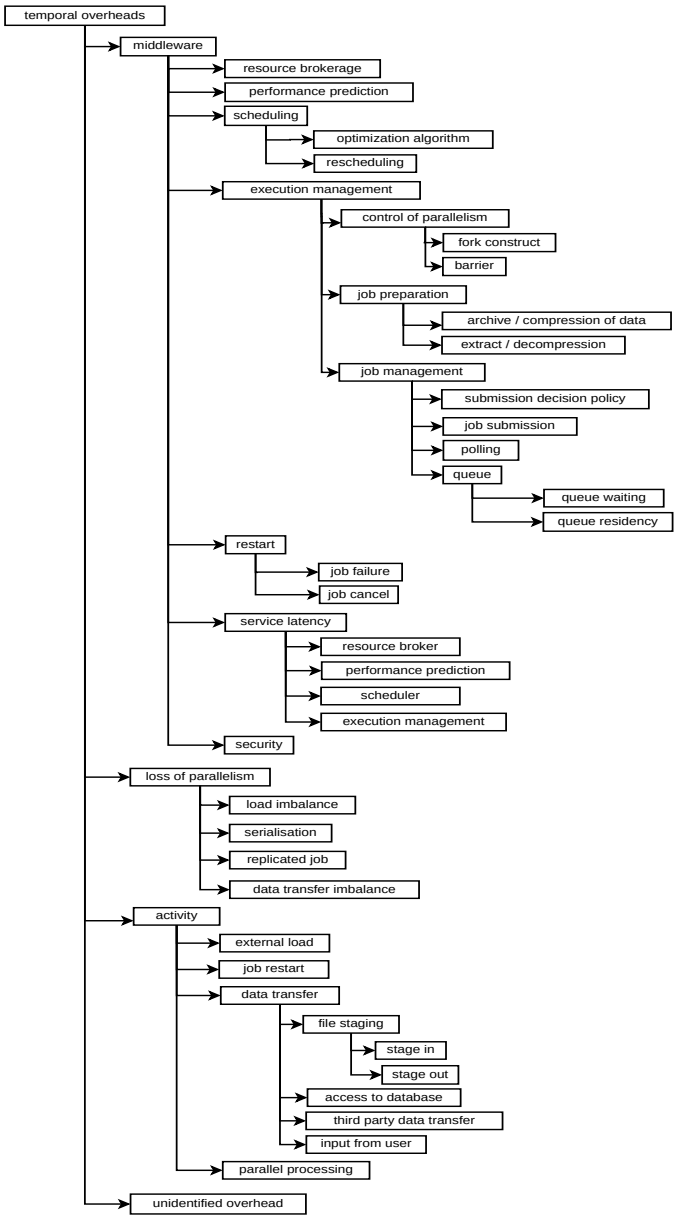



Fig. 1. Grid workflow overhead classification.

formula that expresses the time required to complete a task: $TOS_{MS} = \max_{\forall end(MS)}\{end(MS)\} - \max_{\forall start(MS)}\{start(MS)\}$.

*2) Loss of parallelism overheads:* The first question that arises when computing the parallel section overheads for any workflow region is how we compute them based on the overheads collected for each individual activity.

Let $TO_{CA}$ denote an arbitrary temporal overhead of a computational activity *CA*. We define the temporal overhead $TO_{N_{Par}}$ for a full parallel section $N_{Par}$ as the sum of the contributed overheads $TOcon_{CA}$ of all individual activities $CA \in N_{Par}$. The contributed overhead of an activity *CA* to an enclosing parallel section is the temporal overhead $TO_{CA}$ averaged across the number of processors *used* in executing the parallel section (i.e. as indicated by the Scheduler, which is

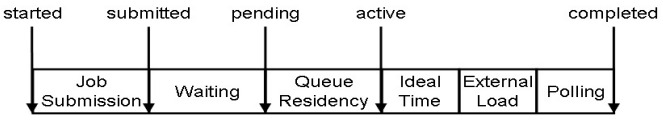



Fig. 2. Activity-related events and middleware overheads.

*not* necessarily the full available Grid), where each processor *P* is weighted with its relative speed (execution time of *CA* on processor *P*): $TO_{N_{Par}} = \sum_{\forall CA \in N_{Par}} TOcon_{CA}$, where $TOcon_{CA} = \frac{TO_{CA}}{\sum_{\forall P \in Grid} \frac{T_{CA}}{T_{CA\_P}}}$ and $T_{CA\_P}$ denotes the wallclock time of the computational activity *CA* on processor *P*. The rationale behind this formula is that, for example, one minute of queuing time before running on a fast Pentium 4 processor is more critical than waiting the same amount of time for Pentium 3 processor and, therefore, the former should produce a proportionally higher temporal overhead value.

In addition, workflow parallel sections introduce a new source of performance overheads that we call *loss of parallelism*. It is common in practice that, as a result of a scheduling algorithm, two parallel computational activities are mapped onto the same (fast) processor that delivers the earliest completion time. This is often due to the fact that the number of fast processors available in our Grid is smaller than the size of the workflow parallel sections. The Scheduler and the Enactment Engine handle this situation by serialising the execution of parallel activities, which is usually faster than competing for processor cycles, memory modules, or other resources.

If two independent *CA* activities (e.g. part of a parallel activity) are scheduled onto the same machine, the Scheduler and the Enactment Engine introduce a *run-time schedule dependency* that prohibits the two activities from running simultaneously on the same processor. We call the set of parallel activities of a parallel section $N_{Par}$ serialised through run-time schedule dependencies as a *serialised block*: $B = \{CA_1, \ldots, CA_n \in N_{Par} | \ \mathcal{S}(CA_1) = \ldots = \mathcal{S}(CA_n)\}$, where $\mathcal{S}(CA)$ denotes the schedule of the task *CA*. The execution time of a serialised block is the sum of the serialised activities: $T_B = \sum_{\forall CA \in B} T_{CA}$.

For parallel computers as Grid sites, we introduce run-time schedule dependencies if the number of parallel activities exceeds the number of available parallel processors. For example, assume in Figure 3 a parallel section that consists of seven parallel activities: $N_{Par} = (CA_1, CA_2, CA_3, CA_4, CA_5, CA_6, CA_7)$ such that: $\mathcal{S}(CA_1) = \mathcal{S}(CA_4) = \mathcal{S}(CA_7) = P_1$, $\mathcal{S}(CA_2) = \mathcal{S}(CA_5) = P_2$, and $\mathcal{S}(CA_3) = \mathcal{S}(CA_6) = P_3$ (where $P_1$ and $P_2$ are part of a dual SMP node of a parallel computer). Since the number of parallel activities exceeds the number of processors available, the

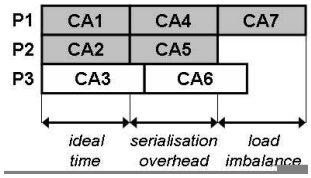



Fig. 3. Parallel section overheads.

Scheduler generates three serialised blocks by enhancing the workflow with the following run-time schedule dependencies: $\{(CA_1, CA_4), (CA_4, CA_7), (CA_2, CA_5), (CA_3, CA_6)\}$.

The *load imbalance* overhead occurs in the context of a parallel section when some of the computational activities finish earlier and leave the allocated processors idle. We define the load imbalance overhead of a parallel activity $N_{Par}$ as the difference between the maximum serialisation block end-time and the ideal execution time of the parallel section on the set of scheduled processors $\mathcal{S}(N_{Par})$ (see Figure 3): $TOLI_{N_{Par}} = \max_{\forall B \in N_{Par}} \{T_B\} - T^{ideal}_{\mathcal{S}(N_{Par})}$. We compute the term $T^{ideal}_{\mathcal{S}(N_{Par})}$ from the following equation: $Work = \sum_{\forall P \in \mathcal{S}(N_{Par})} Work_P = \sum_{\forall P \in \mathcal{S}(N_{Par})} v_P \cdot T^{ideal}_{\mathcal{S}(N_{Par})} = \sum_{\forall P \in \mathcal{S}(N_{Par})} \frac{Work}{T_{seq\_P}} \cdot T^{ideal}_{\mathcal{S}(N_{Par})}$, where *Work* is the total computational work of all activities in the parallel section, $Work_P$ is the ideal distribution of the parallel work on processor *P*, $v_P$ is the speed of processor *P*, and $T_{seq\_P}$ is the sequential execution time of the entire parallel section serialised on the processor *P*. The ideal time becomes therefore the harmonic mean of the single processor sequential times divided by the number of processors: $T^{ideal}_{\mathcal{S}(N_{Par})} = \frac{1}{\sum_{\forall P \in \mathcal{S}(N_{Par})} T^{-1}_{seq\_P}}$.

We define the *data transfer imbalance* in a simpler way, as the difference between the maximum and the average end-time of the parallel file transfers: $TODI_{N_{Par}} = \max_{\forall DA \in N_{Par}} \{end(DA)\} - \operatorname{avg}_{\forall DA \in N_{Par}} \{end(DA)\}$.

Finally, we calculate the *serialisation overhead* of a parallel section as follows (see Figure 3): $TOSER_{N_{Par}} = T_{N_{Par}} - \min_{\forall N \in N_{Par}} \{T_N\} - TOLI_{N_{Par}}$, where the load imbalance overhead in this case has an additional serialisation aspect (in contrast to the case when all activities of a parallel section are executed simultaneously).

Sometimes replicating the same computation on multiple sites in parallel is faster than computing it on the fastest Grid site and then broadcasting the data. Since it does not make sense to synchronise replicated jobs and produce load imbalance, we calculate the *replicated job* overhead of a set of identical activities $N_{Par}$ as the ideal time for computing the entire replicated work (i.e. harmonic mean of all replicated job end-times divided by the number of jobs) minus the end-time of the fastest replicated job representing the only useful computation: $TORJ_{N_{Par}} = \sum_{\forall CA \in N_{Par}} \frac{1}{\sum_{CA \in N_{Par}} T^{-1}_{CA}} - T_{CA_1}$, where $T_{CA_1} \leq T_{CA}, \forall\ CA \in N_{Par}$.

*3) Activity overheads:* We outline in the following the method of computing some of the most important overheads related to individual computational activities.

We define the *external load* overhead of a computational activity *CA* as the difference between the measured wallclock time and its ideal execution time: $TOEL_{CA} = T_{CA} - T^{ideal}_{CA}$.

We compute the *restart* overhead due to a *job failure* of a computational activity *CA* as the time difference between the earliest *start* and the latest *fail* timestamp events: $ORS_{CA} = \max_{\forall fail(CA)} \{fail(CA)\} - \min_{\forall start(CA) < fail(CA)} \{start(CA)\}$.

We define the *data transfer* overhead of an activity *DA* that does a (e.g. GridFTP) file transfer as: $TODT_{DA} = \max_{\forall end(DA)} \{end(DA)\} - \max_{\forall start(DA)} \{start(DA)\}$.

Further, we include in our classification the parallel processing overheads extensively addressed in the previous research.

*4) Unidentified overhead:* Let $T_{WF}$ be the measured execution time, $T^{ideal}_{WF}$ the ideal execution time, $TO^{identified}_{WF}$ the sum of the measured workflow overheads, and $TO_{WF}$ the (theoretical) total overhead introduced in Section III-A. We call the difference between the total overhead and the sum of the identified overheads as *unidentified overhead*: $TO^{unidentified}_{WF} = TO_{WF} - TO^{identified}_{WF}$. Minimising the unidentified overhead is one of the important goals of our performance analysis effort. A high unidentified overhead value indicates that the analysis is unsatisfactory and further efforts are required to spot new sources of overhead in the workflow execution.

### C. Normalised Metrics

Normalised metrics are valuable means for understanding the importance of the temporal overheads with respect to the entire workflow execution.

The value of an overhead $TO$ normalised against the workflow execution time $T$ represents the *overhead severity* which quantifies the importance of the performance overhead for the specific workflow execution: $SEV = \frac{TO}{T}$. The higher the severity for an overhead, the more important or severe this overhead is.

In defining the speedup and efficiency metrics for Grid workflows, we considered to be more realistic to use the Grid site as the sequential computational unit rather than individual processors.

We define the workflow *speedup* as the ratio between the execution time $T_{seq\_M}$ on the fastest (for this workflow) single site available and the actual execution time of the workflow on the Grid $T$: $S = \frac{\min_{\forall M \in Grid} \{T_{seq\_M}\}}{T}$. The rationale behind this formula is that only by normalising against the *fastest* parallel computer available the scientists can understand the potential gain of using the Grid.

Further, we define the workflow *efficiency* as the speedup normalised against the number of the Grid sites used, where each Grid site *M* is weighted with the speedup of the corresponding single site execution time:

$$E = \frac{S}{\sum_{M \in Grid} S_M}, \text{ where: } S_M = \frac{\min_{\forall M' \in Grid} \{T_{seq\_M'}\}}{T_{seq\_M}}.$$

The efficiency formula becomes therefore: $E = \frac{T^{-1}}{\sum_{\forall M \in Grid} T^{-1}_{seq\_M}}$. The fastest Grid site has a weight of one, whereas the slowest Grid site has the smallest weight (i.e. closest to zero).

Finally, since our main interest is on measuring overheads, we define the *inefficiency* as the converse of the efficiency metric: $I = 1 - E$.

## IV. Experiments

Beyond a detailed overhead analysis, our experiments try to answer the following question. Assume that we execute and measure the execution time on the fastest Grid site available

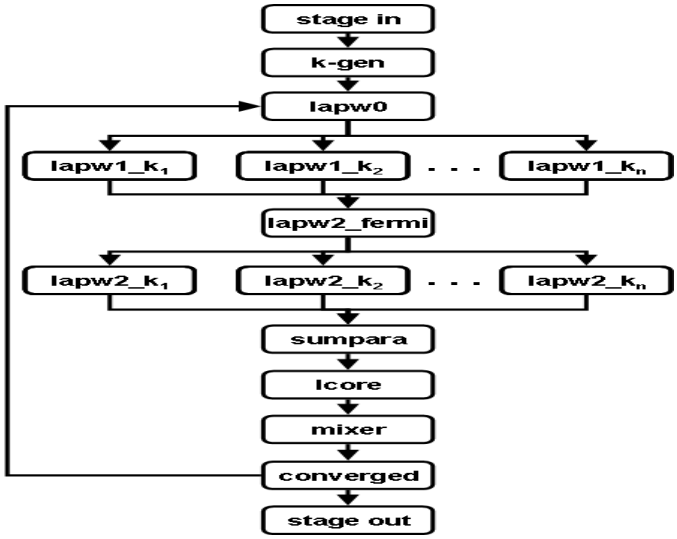



Fig. 4. The WIEN2k workflow.

for a certain workflow. Thereafter, we incrementally add the next fastest site to the Grid execution environment. If the execution time decreases substantially with each additional Grid site, then we demonstrate to the application developers good performance potential for running in a distributed Grid environment.

We have performed experiments for two different workflow applications by varying various parameters and Grid testbeds, aggregating up to 250 processors altogether using Globus Toolkit version 2 as the underlying infrastructure. We have started by executing each workflow application on every individual Grid site available and ranked the sites according to the execution times obtained (see Tables I and II). After establishing a ranking of the Grid sites for every application, we proceeded with the repeated execution of the workflow on multiple Grid sites, by incrementally adding the next fastest site to the Grid environment (in different executions).

### A. WIEN2k

WIEN2k (see *http://www.wien2k.at*) is a program package for performing electronic structure calculations of solids using density functional theory, based on the full-potential (linearised) augmented plane-wave ((L)APW) and local orbitals (lo) method. We have ported the WIEN2k application onto the Grid by splitting the monolithic code into several course-grain activities coordinated in a workflow, as illustrated in Figure 4. The *lapw1* and *lapw2* activities can be solved in a parallel region by a fixed number of so-called *k-points*. A final activity called *Converged* is applied on several output files and tests whether the problem convergence criterion is fulfilled. The number of recursive loops is statically unknown. We have chosen a problem case (called *atype*) that we solved by using two different number of parallel k-points (i.e. size of the two parallel sections – *lapw1* and *lapw2*): 100 and 252.

For the large 252 k-point parallelization, the speedup curve displayed in Figure 5(a) shows that the Grid execution improves for up to seven sites. The improvement comes from the parallel execution of the computationally expensive k-points on multiple Grid sites that significantly lowers the execution time of the parallel sections. The speedup for the 100 k-point parallelization deteriorates beyond four Grid sites, which we will explain in the next paragraphs (see the load imbalance overhead). The efficiency curve slightly decreases in both cases and remains above 0.5, which we find quite promising considering the modest problem sizes that we executed (i.e. overall completion time of about 10, respectively 30 minutes on eight sites) and the rather high overheads experienced, which we describe in the remainder of this section. Figures 5(d) and 5(e) compare the ideal execution time and the most significant overheads against the real Grid execution time. We have marked with asterisks the most severe overheads (the legible ones in these figures), which the reader can follow in a top-bottom order on the individual bars.

The severity of the total overhead for the 100 k-points experiment constantly decreases with the Grid size from over 80% on one site to 45% on eight Grid sites (see Figure 5(b)). Figure 5(c) summarises in one graph the overhead severities in every Grid site configuration, which indicates the importance of each overhead and guides the application and middleware developers on which parts to concentrate their tuning efforts.

The most important overhead is the serialisation overhead due to the limited Grid size, that counts for over 90% of the total overhead on a singe site, but decreases to zero on eight sites. This overhead indicates the performance one could gain by acquiring or adding new processors to the Grid environment. If extending the Grid size is irrelevant, one could consider this overhead as part of the ideal execution time.

The second severe overhead is the loss of parallelism due to load imbalance that fluctuates depending on the number of k-points, processors available, and the size of the serialised block of each processor (see Section III-B.2). For instance, Figure 5(a) shows that the speedup for 100 parallel k-points remains constant for four, five, and six Grid sites, and increases again once we add the seventh site. By adding the fourth site (gescher) to the Grid infrastructure, we reach a number of 58 processors, so we only need serialised blocks of size two to complete the execution of the first parallel section (that is the most time consuming section of the workflow). By adding the fifth and the sixth sites, we reach a total of 78, respectively 98 processors which are *not* enough for executing the parallel

| *Site* | *#* | *CPU, GHz* | *Arch.* | *Mgr.* | *Location* |
|---|---|---|---|---|---|
| altix1.jku | 14 | Itanium2,1.6 | ccNUMA | PBS | Linz |
| altix1.uibk | 14 | Itanium2,1.6 | ccNUMA | PBS | Innsbruck |
| schafberg | 14 | Itanium2,1.6 | ccNUMA | PBS | Salzburg |
| gescher | 16 | Pentium4,3 | COW | Maui | Vienna |
| agrid1 | 20 | Pentium4,1.8 | NOW | Torque | Innsbruck |
| arch19 | 20 | Pentium4,1.8 | NOW | Torque | Innsbruck |
| arch20 | 20 | Pentium4,1.8 | NOW | Torque | Innsbruck |
| arch21 | 20 | Pentium4,1.8 | NOW | Torque | Innsbruck |

TABLE I

GRID TESTBED FOR WIEN2K 100 AND 252 K-POINTS.

section without any serialised block. This produces a high load imbalance overhead which has the severe impact on the speedup that we mentioned before. By adding the seventh site, we gain again speedup since we reach the number of $100$ processors required for eliminating any serialised block that produces the load imbalance.

The next important overhead is the job preparation overhead for compression / decompression of a large number of files into / from an archive, with the purpose of reducing the data transfer overhead. The WIEN2k activities have a large number of data dependencies (i.e. files) that increase proportionally with the number of parallel k-points (about three times). Moreover, the size of the data that needs to be transferred between activities increases with the number of k-points and Grid sites (about $500$ MBytes for $100$ k-points). Therefore, it becomes crucial to archive and compress the output files before transferring them over the network to the next activity. This overhead remains relatively constant for the first four Grid site infrastructures. The last four Pentium 4 workstation networks (which are part of a large, intensively used student workstation network), however, exhibit unexpectedly large access latencies to the shared AFS file system upon decompressing file archives of about $50$ MBytes. This overhead grows linearly with the number of archives used (i.e. $n-1$ tar archives for $n$ Grid sites) that significantly slows down the execution.

We managed to keep the data transfer overhead relatively constant (about $140$ seconds) by using parallel streams over the GridFTP protocol to transfer the archives between sites. Additionally, we exhibit a constant imbalance on parallel data transfers between $50$ and $60$ seconds per workflow execution.

The overheads of the ASKALON middleware services, comprising the Resource Broker, the Scheduler, and the Performance Prediction service are constant and count for less than $1\%$ each of the entire execution time.

Our tool can also perform analysis on a region basis, which is useful in tracing the execution behaviour, for example in dynamic environments like the Grid. To illustrate this, we have run another WIEN2k case consisting of $193$ k-points in a more dynamic Grid testbed illustrated in Table II, where the clusters with the ranks $4-7$ are workstation networks that are heavily used during the day by bachelor students for their exercises (usually in Windows mode), and are automatically rebooted into Linux Grid mode during night, weekend, and public holidays. Figure 5(f) shows the overheads for the four main regions of the WIEN2k workflow, where the last three activities have been merged for performance reasons. Since the starting number of processors was bigger than the number of k-points, in the first parallel region (*lapw1*) we did not experience any serialisation overhead. For this particular first experiment of this kind, we have forced a manual shutdown of the GRAM gatekeeper of the agrid, arch20, arch21, and agrid1 sites as soon as the workflow reached the middle FERMI activity (we will describe a more realistic scenario in Section IV-B). As a consequence, the Enactment Engine rescheduled the *lapw2* activities onto the three remaining SGI Altix Grid sites which produced the serialisation overhead of running multiple parallel activities sequentially due to the lack of enough processors.

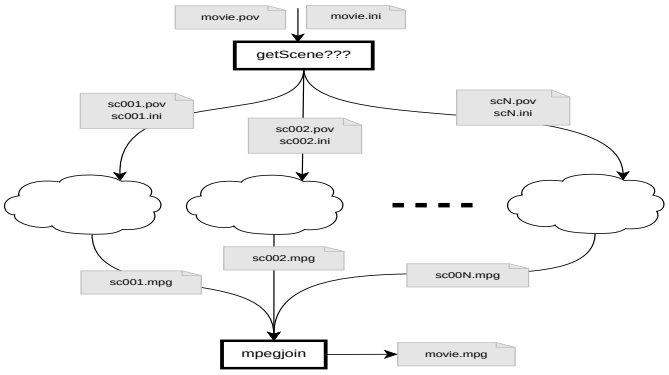



Fig. 6. The POV-Ray workflow.

### B. POV-Ray

The Persistence Of Vision Raytracer (see *http://www.povray.org*) is a high quality free tool for creating three-dimensional graphics, which is known to be an extremely time consuming process. We have modeled a POV-Ray rendering scenario as a workflow depicted in Figure 6, where the description of a movie can be separated in several scenes, each scene being composed of several frame images (e.g. in .png format) which can be rendered as parallel activities on the Grid. Finally, all the frames are merged into a .mpg movie.

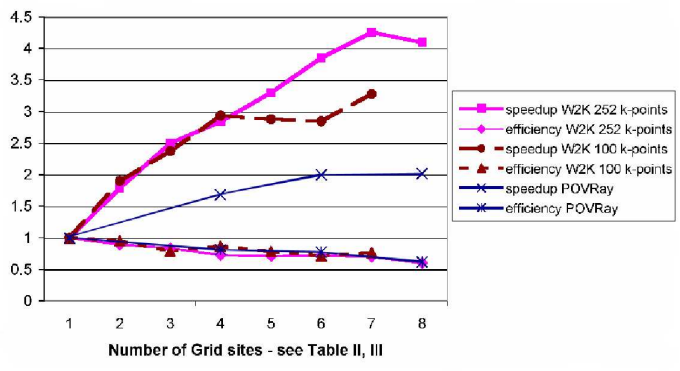
We chose to render a movie from a POV-Ray Internet Raytracing Competition consisting of $19$ scenes and $1650$ frames. Table II displays the Grid testbed that we used for this experiment.

The speedup for this application displayed in Figure 5(a) is lower than for WIEN2k, primarily because of the different Grid testbed used, where the most powerful site is now four times faster than the second site in the ranking (see Table II). For example, from one to four Grid sites did not even double the the number of processors, therefore we only obtained a speedup of $1.69$. The efficiency, however, follows the same good curve as for WIEN2k and stays over $50\%$ for the largest eight site Grid configuration used.

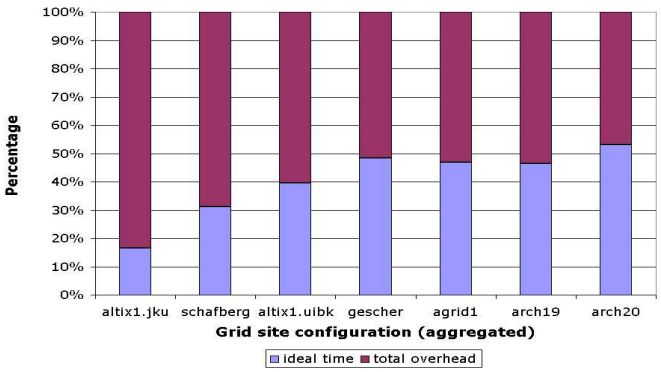
Because the number of workflow activities corresponding to the frames to be rendered is larger than the Grid testbed ($1650$

| *Site* | *#* | *CPU, GHz* | *Arch.* | *Mgr.* | *Location* |
|---|---|---|---|---|---|
| altix.uibk | 64 | Itanium2,1.6 | ccNUMA | PBS | Linz |
| altix.jku | 16 | Itanium2,1.6 | ccNUMA | PBS | Innsbruck |
| schafberg | 16 | Itanium2,1.6 | ccNUMA | PBS | Salzburg |
| agrid | 21 | Pentium4,1.8 | NOW | Torque | Innsbruck |
| arch20 | 23 | Pentium4,1.8 | NOW | Torque | Innsbruck |
| arch21 | 21 | Pentium4,1.8 | NOW | Torque | Innsbruck |
| agrid1 | 32 | Pentium4,1.8 | NOW | Torque | Innsbruck |
| hc-ma | 8 | Opteron,2.2 | COW | SGE | Innsbruck |

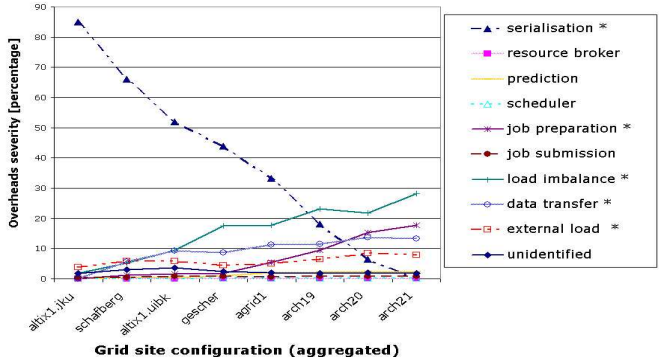
TABLE II
THE GRID TESTBED FOR POV-RAY AND WIEN2K 193 K-POINTS.



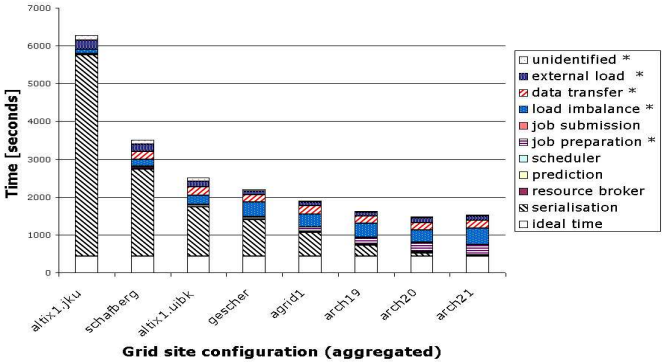

(a) WIEN2k and POV-Ray speedup and efficiency.



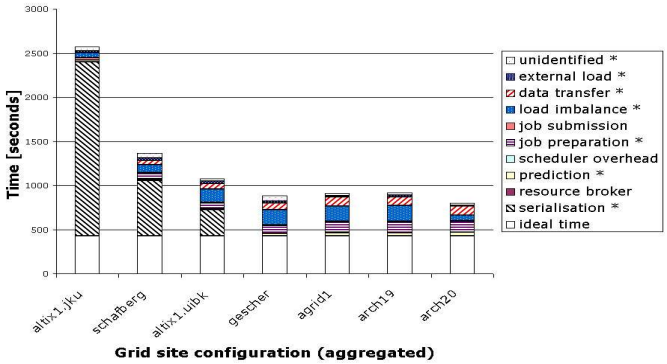

(b) WIEN2k total overhead severity; 100 k-points.



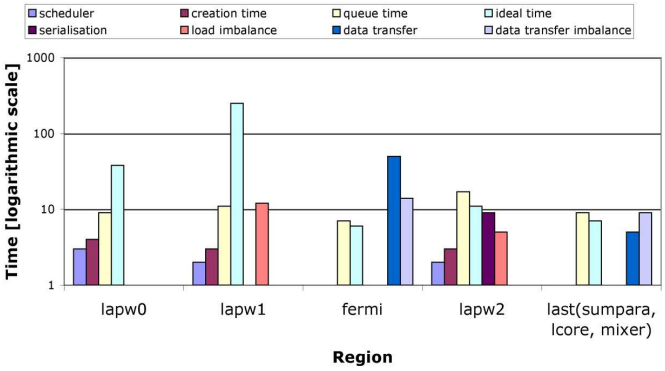

(c) WIEN2k overhead severities; 252 k-points.



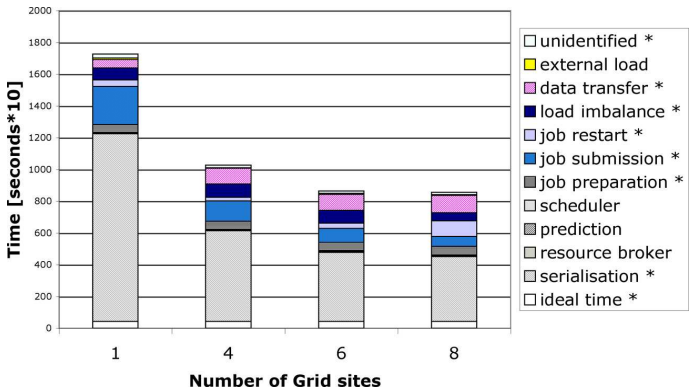

(d) WIEN2k execution overheads; 252 k-points.



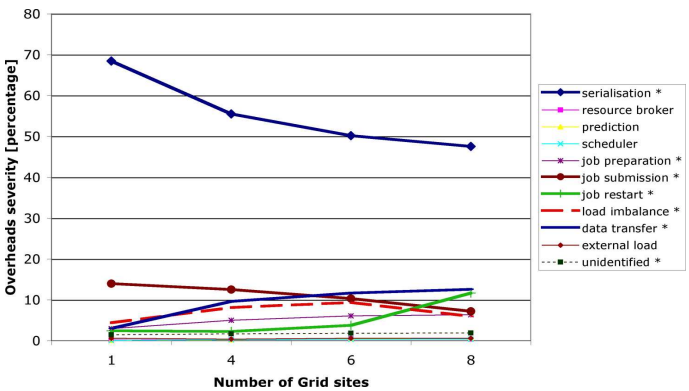

(e) WIEN2k execution overheads; 100 k-points.




(f) WIEN2k region based overheads; 193 k-points; 7 sites.




(g) POV-Ray execution overheads; 1650 frames.




(h) POV-Ray overhead severities; 1650 frames.

Fig. 5. Performance analysis results.

parallel activities versus $201$ processors), the main overhead is again the serialisation (see Figure 5(h)). It is interesting to notice that for this workflow the severity of the job submission overhead is considerable, as a large number of jobs had to be submitted to remote Grid sites.

For the eight site execution, we went again through the same dynamic scenario as for the last WIEN2k experiment, this time more realistically by running the workflow shortly before $8AM$ when the student classes start. A few minutes after $8AM$, the teachers rebooted to Windows mode the master workstations running the GRAM gatekeeper of the student labs ranked $4-8$ in Table II and, as a consequence, these sites were no longer available for the remainder of the execution. The Enactment Engine handles this situation by resubmitting the remaining frames to the remaining four available sites. The job restart overhead increases to about $12\%$ in severity, since it also counts for the rendering computation that has been lost on the rebooted sites, and which has to be repeated on the remaining parallel machines. This also explains the lack of improvement in the speedup from six to eight Grid sites (see Figure 5(a)). In addition, we optimised the GridFTP-based data transfer by collecting the images as soon as they are generated. Therefore, the severity of the data transfer overhead is low despite the large volume of data to be transferred. In addition, such early data collection significantly reduced the job restart overhead in the eight site execution, since all the image files have already been collected by the time the four sites have been rebooted. Only the running rendering jobs had to be restarted on a different site.

## V. Related Work

Workflow monitoring has been extensively studied for many years in the business field. Many techniques have been introduced to study various quality of service and performance models for workflows [2], [3] and to support monitoring and analysis of workflow executions [4], [5]. However, scientific workflows for Grid computing have different requirements that are not addressed in the business field, like scalability (over Grid and workflow sizes), heterogeneity (of Grid parallel computers and computationally intensive activities), and large complex data dependencies.

Numerous performance monitoring and analysis tools have been developed for the Grid [6], [7], [8], however, most of them target low-level resource monitoring or special-purpose middleware services [9] with little attention paid to workflow-level performance analysis.

Clearly, there is a lack of formal specifications of performance overheads and metrics for Grid workflows compared to similar efforts in the parallel processing arena. A high-level and dedicated performance analysis tool, such as the one we proposed in this paper, is missing.

## VI. Conclusions

In this paper, we proposed a systematic approach to performance analysis of Grid workflow applications. We presented a model consisting of a theoretical ideal execution time and a detailed hierarchy of performance overheads that help the application developers understand the sources of bottlenecks that slow down the distributed execution of scientific workflows in heterogeneous Grid infrastructures. We have carefully defined the overheads to be as little overlapping as possible, which gives us important indication of whether any performance loss remained unidentified. We have adjusted well-known normalised metrics from parallel processing to fit the Grid computing scope, like overhead severity, speedup, and efficiency.

We have implemented our approach within the ASKALON programming and computing environment for the Grid. We learned that serialisation of independent activities, load imbalance, job preparation, and data transfer were the most severe overheads for our application that have to be carefully tuned to further improve the speedup and efficiency Grid metrics. The job submission latencies on the Grid can be substantial and thus, may be a major reason for the poor scalability of workflows with relatively fast computational activities.

## Acknowledgement

This research has been supported by the Austrian Science Fund through the SFBF1104 project Aurora and by the European Union through the IST-2002- 511385 project K-Wf Grid.

## References

[1] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek, "ASKALON: A Grid Application Development and Computing Environment," in *6th International Workshop on Grid Computing (Grid 2005)*. Seattle, USA: IEEE Computer Society Press, Nov. 2005.

[2] J. Cardoso, A. P. Sheth, and J. Miller, "Workflow quality of service," in *IFIP TC5/WG5.12 International Conference on Enterprise Integration and Modeling Technique*. Kluwer, B.V., 2003, pp. 303–311.

[3] K.-H. Kim and C. A. Ellis, "Performance analytic models and analyses for workflow architectures," *Information Systems Frontiers*, vol. 3, no. 3, pp. 339–355, 2001.

[4] A. F. Abate, A. Esposito, N. Grieco, and G. Nota, "Workflow performance evaluation through wpql," in *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. ACM Press, 2002, pp. 489–495.

[5] B. T. R. Savarimuthu, M. Purvis, and M. Fleurke, "Monitoring and controlling of a multi-agent based workflow system," in *Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation*. Australian Computer Society, Inc., 2004, pp. 127–132.

[6] M. Gerndt, R. Wismueller, Z. Balaton, G. Gombas, P. Kacsuk, Z. Nemeth, N. Podhorszki, H.-L. Truong, T. Fahringer, M. Bubak, E. Laure, and T. Margalef, *Performance Tools for the Grid: State of the Art and Future*, ser. Research Report Series, Lehrstuhl fr Rechnertechnik und Rechnerorganisation (LRR-TUM) Technische Universitaet Muenchen. Shaker Verlag, 2004, vol. 30.

[7] S. Zanikolas and R. Sakellariou, "A Taxonomy of Grid Monitoring Systems," *Future Generation Computing Systems*, vol. 21, no. 1, pp. 163–188, January 2005.

[8] H.-L. Truong and T. Fahringer, "SCALEA-G: A Unified Monitoring and Performance Analysis System for the Grid," *Scientific Programming*, vol. 12, no. 4, pp. 225–237, January 2004.

[9] X. Zhang, J. L. Freschl, and J. M. Schopf, "A performance study of monitoring and information services for distributed systems," in *International Symposium on High Performance Distributed Computing*. IEEE Computer Society, 2003.