GRID-BASED WORKFLOW MANAGEMENT FOR AUTOMATIC PERFORMANCE ANALYSIS OF MASSIVELY PARALLEL APPLICATIONS

Daniel Becker[1,2], Morris Riedel[1], Achim Streit[1], and Felix Wolf[1,2]

1 *Forschungszentrum Jülich, Institute for Advanced Simulation*
52425 Jülich, Germany
{d.becker, m.riedel, a.streit, f.wolf}@fz-juelich.de

2 *RWTH Aachen University, Department of Computer Science*
52056 Aachen, Germany

Abstract

Many Grid infrastructures have begun to offer services to end-users during the past several years with an increasing number of complex scientific applications and software tools that require seamless access to different Grid resources via Grid middleware during one workflow. End-users of the rather HPC-driven DEISA Grid infrastructure take not only advantage of Grid workflow management capabilities for massively parallel applications to solve critical problems of high complexity (e.g. protein folding, global weather prediction), but also leverage software tools to achieve satisfactory application performance on contemporary massively parallel machines (e.g., IBM Blue Gene/P). In this context, event tracing is one technique widely used by software tools with a broad spectrum of applications ranging from performance analysis, performance prediction and modeling to debugging. In particular, automatic performance analysis has emerged as an powerful and robust instrument to make the optimization of parallel applications both more effective and more efficient. The approach of automatic performance analysis implies multiple steps that can perfectly leverage the workflow capabilities in Grids. In this paper, we present how this approach is implemented by using the workflow management capabilities of the UNICORE Grid middleware, which is deployed on DEISA, and thus, demonstrate by using a Grid application that the approach taken is feasible.

Keywords:

Middleware, workflow management, performance analysis, HPC applications.

1. Introduction

Grid infrastructures offer services to end-users with an increasing number of complex scientific applications and software tools. Due to the increased complexity of contemporary High-Performance Computing (HPC) systems, software tools that are used to tune parallel applications are important. Using these rather complicated software tools can be significant simplified by Grid middleware providing multi-step workflow capabilities. While a wide variety of Grid middleware systems exist (gLite [17], Globus Toolkits [14]) today, only a few systems incorporate massive workflow support, and are also driven by HPC needs.

One of those systems is UNICORE, which is deployed on DEISA [3]. This HPC-driven DEISA Grid infrastructure offers not only Grid workflow management capabilities for massively parallel applications to solve critical problems of high complexity (e.g. protein folding, global weather prediction), but also software tools to achieve satisfactory application performance on contemporary massively parallel machines (e.g., IBM Blue Gene/P).

In the area of HPC, event tracing is a popular technique for software tools with a broad spectrum of applications ranging from performance analysis, performance prediction and modeling to debugging. Event traces are helpful in understanding the performance behavior of message-passing applications since they allow in-depth analysis of communication and synchronization patterns. Graphical trace browsers, such as VAMPIR [20] and Paraver [16], allow the fine-grained investigation of parallel performance behavior and provide statistical summaries. By contrast, automatic trace analysis scans event traces of parallel applications for wait states that occur when processes fail to reach synchronization points in a timely manner, e.g., as a result of an unevenly distributed workload.

The automatic performance analysis approach is a powerful and robust instrument to make the optimization of parallel applications both more effective and more efficient. For instance, SCALASCA, which has been specifically designed for large-scale systems, searches event traces of parallel programs for patterns of inefficient behavior, classifies detected instances by category, and quantifies the associated performance penalty [7]. This allows developers to study the performance of their applications on a higher level of abstraction, while requiring significantly less time and expertise than a manual analysis [23].

SCALASCA's analysis approach implies multiple steps that can perfectly leverage the workflow capabilities in Grids. In this paper, we present how this approach is implemented by using the UNICORE workflow management capabilities that are deployed on DEISA and thus demonstrate by using a Grid application that the approach taken is feasible. While we automize SCALASCA's performance analysis workflow, we show that Grid workflow capabilities in general, and UNICOREs workflow engine in particular, efficiently support the performance analysis process of HPC-driven Grid applications.

The outline of this paper is as follows: After describing the basics of Grid middleware components in Section 2, we review the automatic trace analysis approaches in Section 3 including a brief description of the event tracing technique. In Section 4, we evaluate the feasibility of the approach taken using the state-of-the-art massively parallel machines deployed on the DEISA Grid and show how automatic performance analysis is augmented by UNICORE's workflow capabilities. Finally, after reviewing the related work in Section 5, we summarize our results and give an outlook on future work in Section 6.

2. Grid Middleware

Today, large-scale scientific research often relies on the shared use of a Grid with computational or storage related resources. One of the fundamental ideas of modern Grids is to facilitate the routine interaction of scientists and their workflow-based applications with advanced problem solving tools such as Grid middleware systems. Many of these systems have been evolved in the past, typically influenced by the nature of their deployments. To provide an example, the gLite middleware [17] deployed in the EGEE infrastructure [4] was specifically optimized to handle large data-sets, while the UNICORE middleware [11] deployed in the DEISA infrastructure [3] is rather designed to satisfy the requirements in HPC environments.

Since our approach of supporting the performance analysis of massively parallel applications with Grid-based workflow capabilities relies on a HPC environments, we used key concepts of the HPC-driven Grid middleware UNICORE. Other Grid middleware systems (i.e. Globus Toolkits [14]) may also provide similar capabilities, but the inherently provided workflow functionalities of UNICORE are also fundamental to our approach and thus we choose UNICORE.

The UNICORE Grid middleware has been developed since the late 1990s to support distributed computing applications in Grids in general, and massively parallel HPC applications in particular. The vertically integrated design provides components on each tier of its layered architecture as shown in Figure 1. The UNICORE 5 architecture [22] mainly consists of proprietary components and protocols, while the more recently developed UNICORE 6 architecture is based on open standards such as the Web services resource framework (WS-RF) [5] and thus relies on the concept of Service-Oriented Architectures (SOAs).

In more detail, UNICORE 6 conforms to the Open Grid Services Architecture (OGSA) of the Open Grid Forum (OGF) and several projects such as the OMII-Europe project [6] augmented it with open standards during the last couple of month. The loosely coupled Web services connection technology provides a perfect base to meet the common use case within Grid that conform to OGSA thus allowing dynamic interaction dispersed in conceptual, institutional, and geographical space. The approach presented in this paper takes advantage of this service-oriented concept by using key characteristics of the UNICORE Grid middleware. These characteristics are basic job submission and data management functionalities using the UNICORE Atomic Services (UAS) [21], and the workflow capabilities of the UNICORE workflow engine and service orchestrator.

The UAS consists of a set of core services such as the TargetSystemService (TSS) that represents a computational resource (e.g. supercomputers or clusters) in the Grid. While the TargetSystemFactory (TSF) can be used to create an end-user specific instance of a TSS, the TSS itself is able to execute jobs defined in the Job Submission and Description Language (JSDL) [12]. Each submitted job is represented as another resource aligned with the TSS and can be controlled with the Job Management Service (JMS). Finally, different flavors of the FileTransferServices (FTS) can be used for data transfer.

While the UAS and its services (i.e. TSS, JMS, FTS, etc.) operate on the service level, they are supported by a strong execution backend named as the enhanced Network Job Supervisor (XNJS), which uses the Incarnation Database (IDB) to map abstract job descriptions to system-specific definitions. In terms of security, the UAS are protected via the UNICORE Gateway [18], which does authentication. This means it checks whether the certificate of an end-user has been signed by a trusted Certificate Authority (CA), that its still valid, and not revoked. For authorization the NJS, relies on the UNICORE User DataBase (UUDB) service and checks policies based on Extensible Access Control Markup Language (XACML) [19]. The roles and general attributes of end-users as well as their group and project membership are encoded using Security Assertion Markup Language (SAML) [13] assertions.

Typically, scientists use the UNICORE middleware to reduce their management overheads in usual scientific workflows to a minimum or to automize parts of their workflow. Without Grid middleware, the workflows of scientists often start with the manual creation of a Secure Shell (SSH) connection to a remote system such as a supercomputer via username and password or keys configured before. The trend towards service-oriented Grids allows for more flexibility and orchestration of services and thus makes it easier to enable end-users with multi-step workflow capabilities. While there are many workflow-related tools in Grids (e.g. TAVERNA [8], ASKALON [2], TRIANA [10]), we shortly describe the functionalities of UNICORE that are later intensively used to verify our approach.

To satisfy scalability requirements, the overall workflow capabilities of UNICORE are developed using a two layer architecture. The Workflow Engine is based on the Shark open-source XPDL engine. In addition, plug-ins allow for domain-specific workflow language support. The Workflow engine is responsible to take Directed Acyclic Graph (DAG)-based workflow descriptions and map them to specific steps.

The Service Orchestrator on the other hand, is responsible for the job submission and monitoring of workflow steps. While the execution site can be chosen manually, the orchestrator also supports brokering based on pluggable strategies. In addition, it supports callback functions to the workflow engine.

Finally, all the above described services can be seamlessly accessed using the graphical UNICORE Rich Client based on Eclipse. End-users are able to create DAGs of different services that typically invoke different applications including necessary data staging elements. In addition, the Tracing Service of UNICORE allows for monitoring of each execution step of a UNICORE workflow, which can be easily followed in the client.

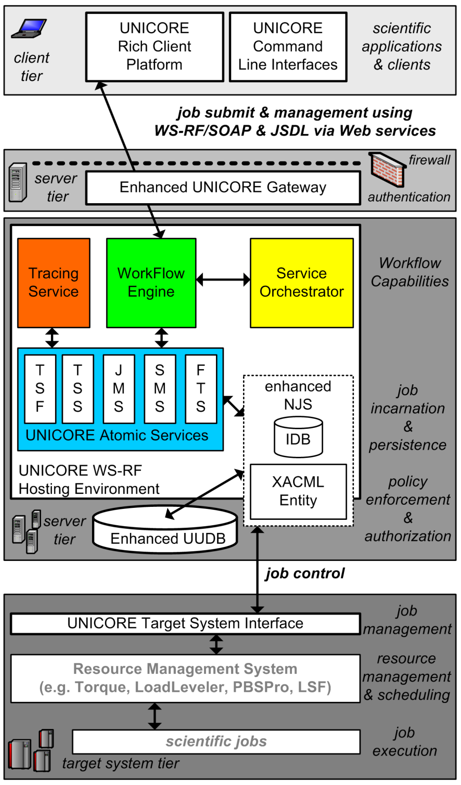

client tier
UNICORE Rich Client Platform
UNICORE Command Line Interfaces
scientific applications & clients
job submit & management using WS-RF/SOAP & JSDL via Web services
firewall
server tier
Enhanced UNICORE Gateway
authentication
Tracing Service
WorkFlow Engine
Service Orchestrator
Workflow Capabilities
TSF
TSS
JMS
SMS
FTS
UNICORE Atomic Services
enhanced NJS
IDB
job incarnation & persistence
UNICORE WS-RF Hosting Environment
XACML Entity
policy enforcement & authorization
server tier
Enhanced UUDB
job control
UNICORE Target System Interface
job management
Resource Management System (e.g. Torque, LoadLeveler, PBSPro, LSF)
resource management & scheduling
scientific jobs
job execution
target system tier

Figure 1. UNICORE's layered architecture with workflow functionalities.

3. Automatic Performance Analysis

In this section, we illustrate the automatic performance analysis workflow including instrumentation, measurement, analysis, and result visualization used to optimize parallel applications running on thousands of processes. After a general description of the event tracing technique and a brief description of the graphical trace browser VAMPIR [20], we especially focus on the SCALASCA tool set [7] and its workflow requirements.

Often, parallel applications which are free of computational errors need to be optimized. This requires the information which component of the program is responsible for what kind of inefficient behavior. Performance analysis is the process of identifying those parts, exploring the reasons for their unsatisfactory performance, and quantifying their overall influence. To do this, performance data are mapped onto program entities. A developer can now investigate application's runtime behavior using software tools. Thus, the developer is enabled to understand the performance behavior of his application. The process of gathering performance data is called performance measurement and forms the basis for subsequent analysis.

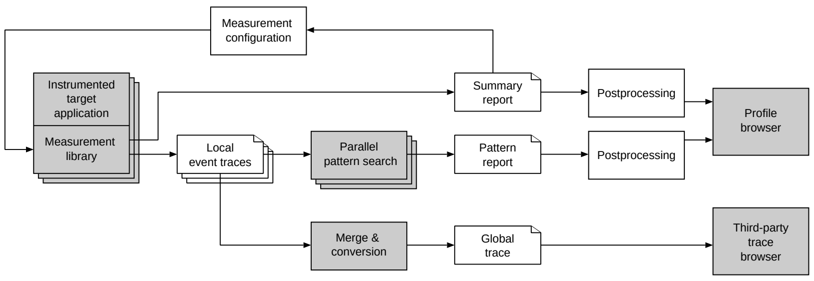


Figure 2. SCALASCA's performance analysis process.

Event tracing is one technique widely used for post-mortem performance analysis of parallel applications. Time-stamped events, such as entering a function or sending a message, are recorded at runtime and analyzed afterwards with the help of software tools. The information recorded for an event includes at least a time stamp, the location (e.g., the process or node) where the event happened and the event type. Depending on the type, additional information may be supplied, such as the function identifier for function call events. Message event records typically contain details about the message they refer to (e.g., the source or destination location and message tag).

Graphical trace browsers, such as VAMPIR, allow the fine-grained, manual investigation of parallel performance behavior using a zoomable time-line display and provide statistical summaries of communication behavior. However, in view of the large amounts of data generated on contemporary parallel machines, the depth and coverage of the visual analysis offered by a browser is limited as soon as it targets more complex patterns not included in the statistics generated by such tools.

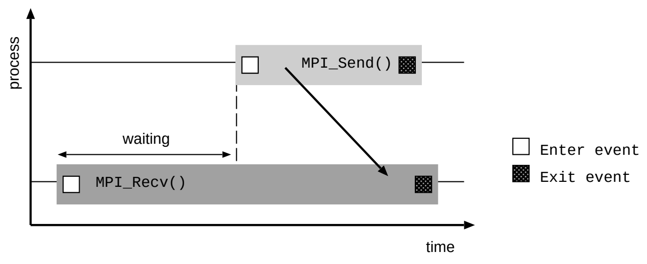


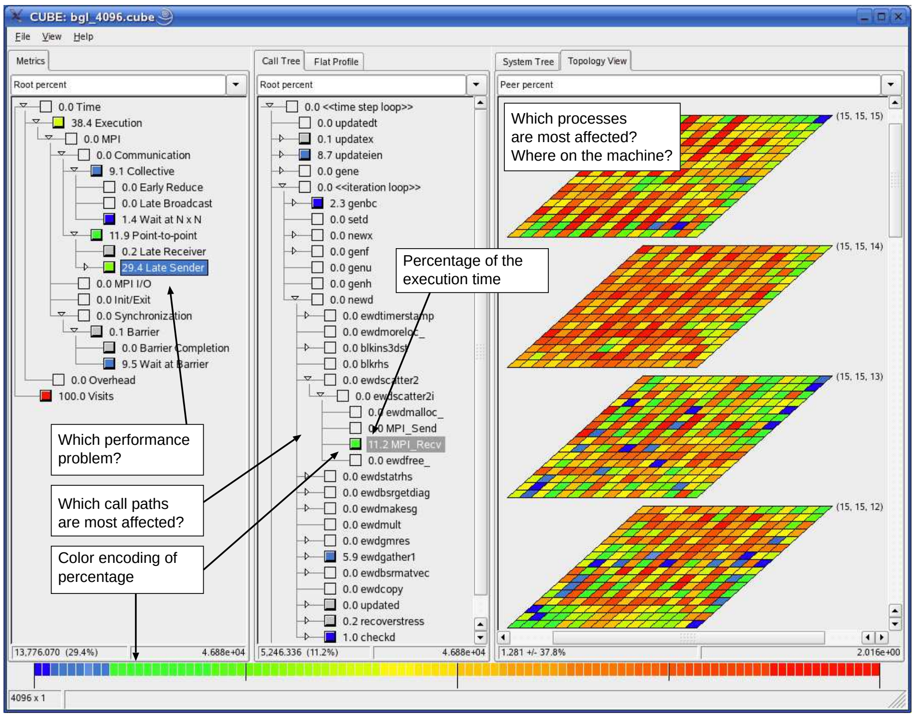
Figure 3. Late Sender pattern.

By contrast, the basic principle of the SCALASCA project is the summarization of events, that is, the transformation of an event stream into a compact representation of execution behavior, aggregating metric values associated with individual events across the entire execution. On a more technical level, the trace analyzer of the SCALASCA tool set automatically searches event traces for patterns of inefficient behavior, classifies detected instances by category, and quantifies the associated performance penalty [15]. To do this efficiently at larger scales, the traces are analyzed in parallel by replaying the original communication using the same hardware configuration and the same number of CPUs as have been used to execute the target application itself.

As an example of inefficient communication, we consider the point-to-point pattern *Late Sender* (Figure 3). Here, a receive operation is entered by one process before the corresponding send operation has been started by the other. The time lost waiting due to this situation is the time difference between the enter events of the two MPI function instances that precede the corresponding send and receive events.

The current version of SCALASCA can be used for MPI programs written in C/C++ and Fortran. Figure 2 shows the basic analysis workflow supported by SCALASCA. Before any performance data can be collected, the target application must be *instrumented*, that is, it must be modified to record performance-relevant events whenever they occur. On some systems including Blue Gene, this can be done completely automatically using compiler support; on other systems a mix of manual and automatic instrumentation mechanisms is offered.

When running the instrumented code on the parallel machine first a summary report (aka profile) with aggregate performance metrics for individual function call paths is generated and subsequently event traces are generated to record individual runtime events. The former is useful to obtain an overview of the



```
CUBE: bgl_4096.cube
File  View  Help

Metrics                          Call Tree | Flat Profile              System Tree | Topology View
Root percent                     Root percent                          Peer percent

0.0 Time                         0.0 <<time step loop>>
  38.4 Execution                   0.0 updatedt
    0.0 MPI                        0.1 updatex
      0.0 Communication            8.7 updateien
        9.1 Collective             0.0 gene
          0.0 Early Reduce         0.0 <<iteration loop>>
          0.0 Late Broadcast         2.3 genbc
          1.4 Wait at N x N          0.0 setd
        11.9 Point-to-point          0.0 newx
          0.2 Late Receiver          0.0 genf
          29.4 Late Sender           0.0 genu
      0.0 MPI I/O                    0.0 genh
      0.0 Init/Exit                  0.0 newd
      0.0 Synchronization              0.0 ewdtimerstamp
        0.1 Barrier                    0.0 ewdmoreloc_
          0.0 Barrier Completion       0.0 blkins3ds
          9.5 Wait at Barrier          0.0 blkrhs
  0.0 Overhead                         0.0 ewdscatter2
100.0 Visits                             0.0 ewdscatter2i
                                           0.0 ewdmalloc_
                                           0.0 MPI_Send
                                           11.2 MPI_Recv
                                           0.0 ewdfree_
                                       0.0 ewdstatrhs
                                       0.0 ewdbsrgetdiag
                                       0.0 ewdmakesg
                                       0.0 ewdmult
                                       0.0 ewdgmres
                                       5.9 ewdgather1
                                       0.0 ewdbsrmatvec
                                       0.0 ewdcopy
                                       0.0 updated
                                       0.2 recoverstress
                                       1.0 checkd

13,776.070 (29.4%)   4.688e+04   5,246.336 (11.2%)   4.688e+04   1.281 +/- 37.8%   2.016e+00
4096 x 1
```

Which processes are most affected? Where on the machine?
(15, 15, 15)
(15, 15, 14)
(15, 15, 13)
(15, 15, 12)
Percentage of the execution time
Which performance problem?
Which call paths are most affected?
Color encoding of percentage

Figure 4. Exemplary trace analysis report: The tree in the left panel displays pattern of inefficient performance behavior arranged in a specialization hierarchy. The numbers left of the pattern names indicate the total execution time penalty in percent. In addition, the color of the little square provides a visual clue of the percentage to quickly guide the user to the most severe performance problems. The middle tree shows the distribution of the selected pattern across the call tree. Finally, the right tree shows the distribution of the selected pattern at the selected call path across the hierarchy of machines, nodes, and processes.

performance behavior and also to optimize the instrumentation for later trace generation. Since traces tend to become very large, this step is crucial before starting event tracing. When tracing is enabled, each process generates a trace file containing records for all its process-local events.

After performance measurement, SCALASCA loads these trace files into main memory and analyzes them in parallel using as many CPUs as have been used for the target application itself. During the analysis, SCALASCA searches for characteristic patterns indicating wait states and related performance properties, classifies detected instances by category and quantifies their significance for every function-call path and system resource involved. The result is a pattern-analysis report similar in structure to the summary report but enriched with higher-level performance metrics.

After performance analysis, both the summary report as well as the pattern report can be interactively explored in a graphical profile browser (Figure 4) after a postprocessing phase. As an alternative to the automatic search, the event traces can be converted and investigated using third-party trace browsers such as Paraver [16] or VAMPIR [20], taking advantage of their powerful time-line visualizations and rich statistical functionality.

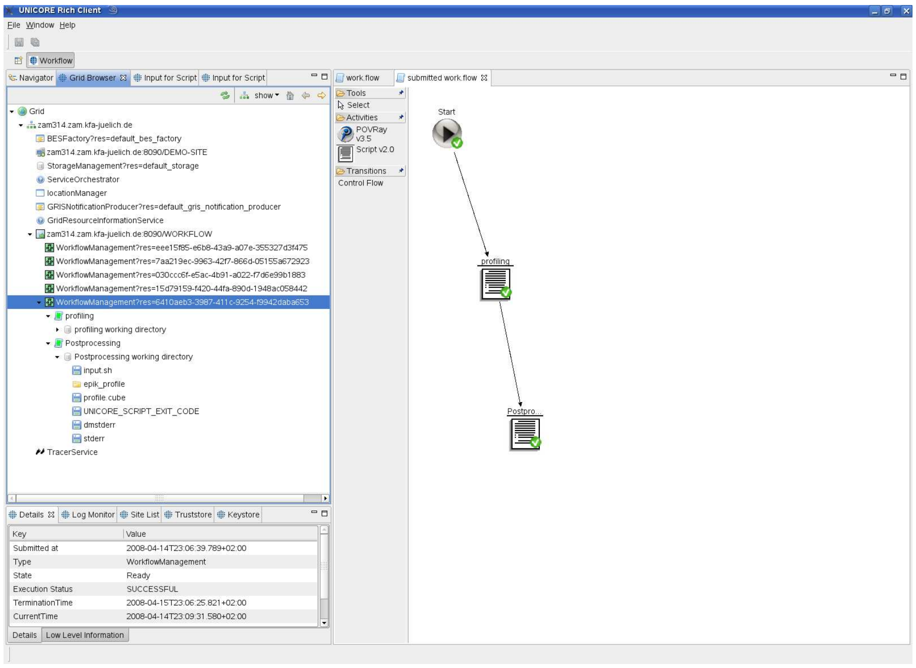
4. Experimental Results

In this section, we present the experimental results of our approach of supporting the performance analysis of massively parallel applications with Grid-based workflow capabilities. Automizing SCALASCA's performance analysis workflow, we show that Grid middleware components efficiently support SCALASCA's pattern search. Finally, we point out that the management overhead is reduced to a minimum and explain the basic performance analysis results detected by the automatic perform.

For our measurements, we used a second generation IBM Blue Gene system, the Blue Gene/P (JUGENE) installed at Research Centre Jülich. The massively parallel JUGENE system is a 16 racks system including 32 nodecards with 32 compute nodes each. While each compute node has a 4-way SMP processor, each core is a 32-bit PowerPC processor core running at 850 MHz. The network architecture of the Blue Gene/P is very similar to that of the L model. That is, the network exhibits a three-dimensional torus, a global tree, and a 10 Gigabit Ethernet network. Ranked as no. 2 in the current Top500 list [9], JUGENE is one of the most powerful high-performance computing systems in the world and especially designed to run massively parallel application.

To evaluate UNICORE's workflow capabilities, we analyzed the performance of the ASC SMG2000 benchmark, a parallel semi-coarsening multigrid solver that uses a complex communication pattern and performs a large number of non-nearest-neighbor point-to-point communication operations. Applying a weak scaling strategy, a fixed $64 \times 64 \times 32$ problem size per process with five solver iterations was configured. In our measurements, we used 512 processes per program run as a prove of concept, while typically a production run uses significant more processors.

Figure 5 shows a snapshot of the UNICORE Rich Client Platform. On the left side of the GUI, we can see the Grid browser showing the actually available services of the infrastructures as well as the status of certain activities. On the right side, we see the runtime summarization of the entire performance analysis workflow. In more detail, both the profiling phase and postprocessing phase (see Figure 2) of the SMG2000 benchmark parallel application is shown. While profiling is enabled, the execution of the pre-compiled and automatically instrumented executable generates a runtime summary. This runtime summary is



```
Grid
  zam314.zam.kfa-juelich.de
    BESFactory?res=default_bes_factory
    zam314.zam.kfa-juelich.de:8090/DEMO-SITE
    StorageManagement?res=default_storage
    ServiceOrchestrator
    locationManager
    GRISNotificationProducer?res=default_gris_notification_producer
    GridResourceInformationService
    zam314.zam.kfa-juelich.de:8090/WORKFLOW
      WorkflowManagement?res=eee15f85-e6b8-43a9-a07e-355327d3f475
      WorkflowManagement?res=7aa219ec-9963-42f7-866d-05155a672923
      WorkflowManagement?res=030ccc6f-e5ac-4b91-a022-f7d6e99b1883
      WorkflowManagement?res=15d79159-f420-44fa-890d-1948ac058442
      WorkflowManagement?res=6410aeb3-3987-411c-9254-f9942daba653
        profiling
          profiling working directory
        Postprocessing
          Postprocessing working directory
            input.sh
            epik_profile
            profile.cube
            UNICORE_SCRIPT_EXIT_CODE
            dmstderr
            stderr
    TracerService
```

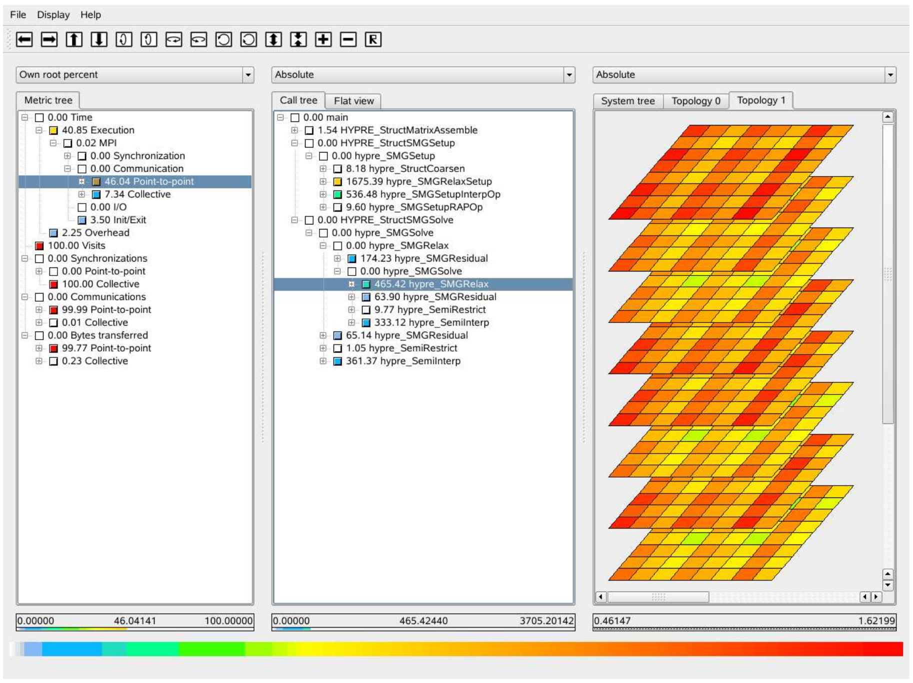
Key	Value
Submitted at	2008-04-14T23:06:39.789+02:00
Type	WorkflowManagement
State	Ready
Execution Status	SUCCESSFUL
TerminationTime	2008-04-15T23:06:25.821+02:00
CurrentTime	2008-04-14T23:09:31.580+02:00

Figure 5. Using UNICORE's workflow capabilities for profiling a parallel application.

subsequently postprocessed via certain preconfigured mappings to be displayed properly to the end-user as shown in Figure 6.

The strength of this runtime summarization is that it avoids storing the events in trace buffers and files, since many execution performance metrics can be most efficiently calculated by accumulating statistics during measurement, avoiding the cost of storing them with events for later analysis. For example, the point-to-point communication time can be immediately determined and finally displayed as shown in Figure 6. Here, the runtime summary report shows a large point-to-point communication fraction distributed across the call tree and the system hierarchy. Obviously, the point-to-point communication time clearly dominated the overall communication behavior making it the most promising target for further analysis efforts. That is, we configured the measurement system to record especially those execution phases related to point-to-point communication and adjusted the buffer sizes, manually.

The UNICORE Rich Client Platform snapshot in Figure 7 shows again the Grid browser with the actually available services of the infrastructures as well as the status of the submitted tracing workflow. This tracing workflow consists of a tracing, trace analysis, and postprocessing phase. During the tracing phase, the measurement system generates process-local event traces maintained in an appropriate measurement archive (`epik_tracing`). The subsequent trace



```
0.00 Time
  40.85 Execution
    0.02 MPI
      0.00 Synchronization
      0.00 Communication
        46.04 Point-to-point
        7.34 Collective
      0.00 I/O
      3.50 Init/Exit
    2.25 Overhead
  100.00 Visits
0.00 Synchronizations
  0.00 Point-to-point
  100.00 Collective
0.00 Communications
  99.99 Point-to-point
  0.01 Collective
0.00 Bytes transferred
  99.77 Point-to-point
  0.23 Collective

0.00000          46.04141        100.00000
```

```
0.00 main
  1.54 HYPRE_StructMatrixAssemble
  0.00 HYPRE_StructSMGSetup
    0.00 hypre_SMGSetup
      8.18 hypre_StructCoarsen
      1675.39 hypre_SMGRelaxSetup
      536.48 hypre_SMGSetupInterpOp
      9.60 hypre_SMGSetupRAPOp
  0.00 HYPRE_StructSMGSolve
    0.00 hypre_SMGSolve
      0.00 hypre_SMGRelax
        174.23 hypre_SMGResidual
        0.00 hypre_SMGSolve
          465.42 hypre_SMGRelax
          63.90 hypre_SMGResidual
          9.77 hypre_SemiRestrict
          333.12 hypre_SemiInterp
      65.14 hypre_SMGResidual
      1.05 hypre_SemiRestrict
      361.37 hypre_SemiInterp

0.00000          465.42440        3705.20142
```

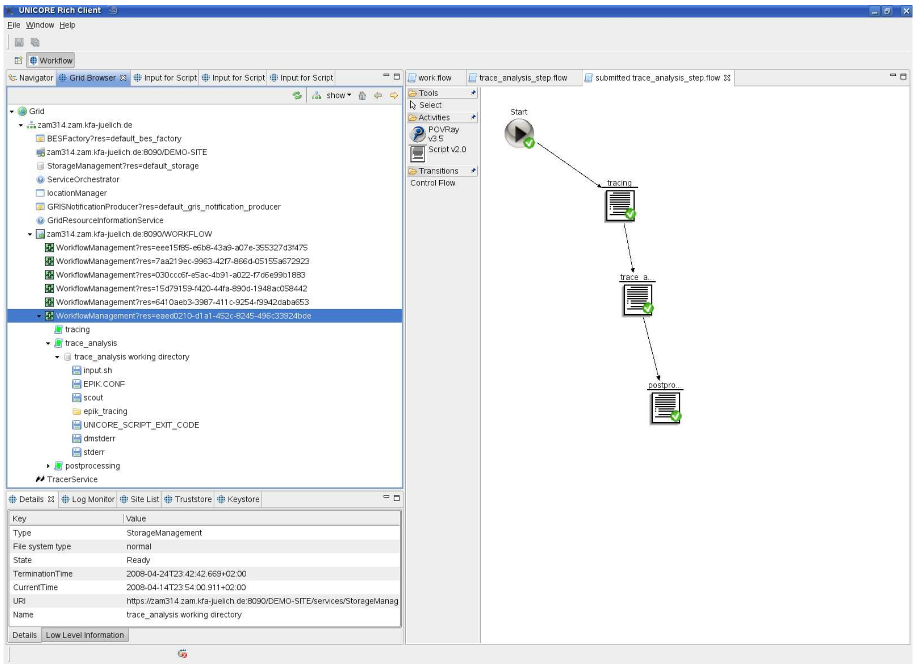
```
0.46147                              1.62199
```

Figure 6. Runtime summary report: Large point-to-point communication fraction distributed across the call tree and the system hierarchy.

analysis starts the parallel trace analyzer, which automatically search those event traces for patterns of inefficient performance behavior. The trace analyzer generates a trace analysis report which is finally postprocessed via certain preconfigured mappings to be displayed properly to the end-user as shown in Figure 8.

SCALASCA searches event traces of parallel programs for patterns of inefficient behavior (e.g., wait states), classifies detected instances by category, and quantifies the associated performance penalty. Often, reasons for such wait states can be found in the scheduling of communication operations or in the distribution of work among the processes involved. Figure 8 shows the trace analysis report similar in structure to the summary report but enriched with higher-level performance metrics. The report indicates that the large point-to-point communication time manifests itself in a Late Sender situation distributed across the call tree and the system hierarchy. Hence, the point-to-point communication would be a prommissing target for performance optimization.

Finally, typical Grid applications execute their workflow steps on different systems, while in our approach all steps (i.e. measurements) are taken on the JUGENE system. This is due to the fact that we analyze the performance for one



```
Grid
  zam314.zam.kfa-juelich.de
    BESFactory?res=default_bes_factory
    zam314.zam.kfa-juelich.de:8090/DEMO-SITE
    StorageManagement?res=default_storage
    ServiceOrchestrator
    locationManager
    GRISNotificationProducer?res=default_gris_notification_producer
    GridResourceInformationService
    zam314.zam.kfa-juelich.de:8090/WORKFLOW
      WorkflowManagement?res=eee15f85-e6b8-43a9-a07e-355327d3f475
      WorkflowManagement?res=7aa219ec-9963-42f7-866d-05155a672923
      WorkflowManagement?res=030ccc6f-e5ac-4b91-a022-f7d6e99b1883
      WorkflowManagement?res=15d79159-f420-44fa-890d-1948ac058442
      WorkflowManagement?res=6410aeb3-3987-411c-9254-f9942daba653
      WorkflowManagement?res=eaed0210-d1a1-452c-8245-496c33924bde
        tracing
        trace_analysis
          trace_analysis working directory
            input.sh
            EPIK.CONF
            scout
            epik_tracing
            UNICORE_SCRIPT_EXIT_CODE
            dmstderr
            stderr
        postprocessing
    TracerService
```

Key	Value
Type	StorageManagement
File system type	normal
State	Ready
TerminationTime	2008-04-24T23:42:42.669+02:00
CurrentTime	2008-04-14T23:54:00.911+02:00
URI	https://zam314.zam.kfa-juelich.de:8090/DEMO-SITE/services/StorageManag
Name	trace_analysis working directory

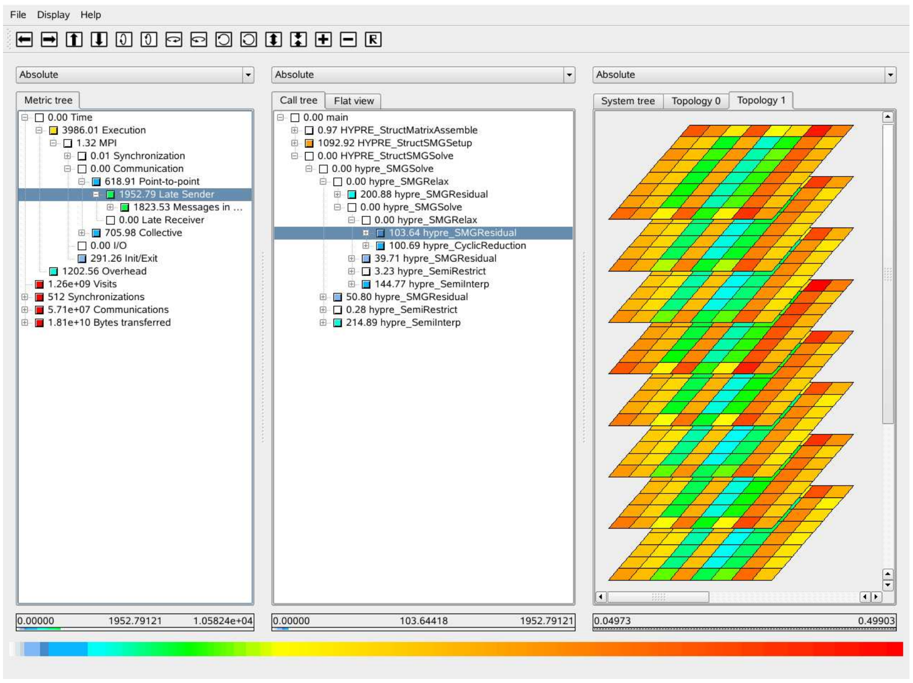
Figure 7. Using UNICORE's workflow capabilities for tracing a parallel application.

dedicated scientific HPC application on a distinct parallel machine. However, using the same workflow on another system is easy to accomplish by just choose another site within the UNICORE Rich Client Platform. Of course, the workflow capabilities cover not only multi-step workflows, but also multi-site workflows that are not substantial to accomplish the performance analysis in this context.

5. Related Work

There is a wide variety of related work in the field. First and foremost, Taverna [8] enables the interoperation between databases and tools by providing a toolkit for composing, executing and managing workflow applications. It's specifically designed to access local and remote resources as well as analysis tools. In comparison to our worfklow approach, this tool is not seamlessly integrated in a Grid middleware.

Another well-known workflow tool named as ASKALON [2]. The fundamental goal of ASKALON is to provide simple, efficient, and effective application development for the Grid. In focusses on workflows as well as parameter studies. It also allows for measurement, analysis and optimization of performance, but also this tool is not closely aligned with HPC environments or any HPC-driven Grid middleware.



```
⊟ 0.00 Time
  ⊟ 3986.01 Execution
    ⊟ 1.32 MPI
      ⊞ 0.01 Synchronization
      ⊟ 0.00 Communication
        ⊟ 618.91 Point-to-point
          ⊟ 1952.79 Late Sender
            ⊞ 1823.53 Messages in ...
            0.00 Late Receiver
        ⊞ 705.98 Collective
      0.00 I/O
      291.26 Init/Exit
    1202.56 Overhead
  1.26e+09 Visits
⊞ 512 Synchronizations
⊞ 5.71e+07 Communications
⊞ 1.81e+10 Bytes transferred
```

```
⊟ 0.00 main
  ⊞ 0.97 HYPRE_StructMatrixAssemble
  ⊞ 1092.92 HYPRE_StructSMGSetup
  ⊟ 0.00 HYPRE_StructSMGSolve
    ⊟ 0.00 hypre_SMGSolve
      ⊟ 0.00 hypre_SMGRelax
        ⊞ 200.88 hypre_SMGResidual
        ⊟ 0.00 hypre_SMGSolve
          ⊟ 0.00 hypre_SMGRelax
            ⊞ 103.64 hypre_SMGResidual
            ⊞ 100.69 hypre_CyclicReduction
          ⊞ 39.71 hypre_SMGResidual
          ⊞ 3.23 hypre_SemiRestrict
          ⊞ 144.77 hypre_SemiInterp
      ⊞ 50.80 hypre_SMGResidual
      ⊞ 0.28 hypre_SemiRestrict
      ⊞ 214.89 hypre_SemiInterp
```

Figure 8. Trace analysis report: Large point-to-point waiting time manifests itself in a Late Sender situation distributed across the call tree and the system hierarchy.

Also, Triana [10] is a problem solving environment especially designed as a Grid workflow tool. Developed by Cardiff University, Triana basically abstracts the core capabilities needed for service-based computing such as P2P, Web services, or Grid Computing. While this approach is rather more oriented to HPC in a way, we also find that the direct support for a Grid middleware was missing.

Finally, also the A-Ware project [1] develops several workflow capabilities specifically designed for service invocation of any kind. Compared to our approach, this work is rather high-level and thus not directly usable for HPC-based environments.

6. Conclusion

In this paper, we have shown that the approach of supporting the performance analysis of parallel HPC applications can be significantly supported by workflow capabilities within Grids. We have mapped the SCALASCA's automatic performance analysis approach of massively parallel applications to UNICORE's workflow management capabilities. More precisely, by automizing SCALASCA's performance analysis workflow, we have shown that Grid middleware components simplify the overall process of SCALASCA's pattern search even when executed on a single Grid resource. Finally, we have explained the basic performance analysis results detected by the automatic performance analysis.

Acknowledgments

The work described in this paper is based on work of many talented colleagues in the UNICORE community that developed UNICORE 6. In particular, we acknowledge our colleagues Bernd Schuller and Bastian Demuth of the Chemomentum project for developing the workflow capabilities within UNICORE 6.

References

[1] A-WARE Project. http://www.a-ware.org/.

[2] ASKALON. http://www.dps.uibk.ac.at/projects/askalon/.

[3] DEISA. http://www.deisa.org/.

[4] EGEE. http://www.eu-egee.org/.

[5] OASIS - WSRF Technical Committee. `http://www.oasis-open.org/committees/tc\_home.php?wg\_abbrev=wsrf`.

[6] OMII - Europe. http://omii-europe.org/.

[7] Scalasca. `www.scalasca.org`.

[8] TAVERNA. http://taverna.sourceforge.net/.

[9] Top500 project. `www.top500.org/`.

[10] TRIANA grid workflow. http://www.grid.org.il.

[11] UNICORE grid middleware. http://www.unicore.eu.

[12] A. Anjomshoaa, M. Drescher, D. Fellows, S. McGougha, D. Pulsipher, and A. Savva. *Job Submission Description Language (JSDL) - Specification Version 1.0*. Open Grid Forum Proposed Recommendation, 2006.

[13] S. Cantor, J. Kemp, R. Philpott, and E. Maler. *Assertions and Protocols for the OASIS Security Assertion Markup Language*. OASIS Standard, 2005. http://docs.oasis-open.org/security/saml/v2.0/.

[14] I. Foster. Globus Toolkit version 4: Software for Service-Oriented Science. In *Proceedings of IFIP International Conference on Network and Parallel Computing, LNCS 3779*, pages 213–223. Springer-Verlag, 2005.

[15] M. Geimer, F. Wolf, B. J. N. Wylie, and B. Mohr. Scalable parallel trace-based performance analysis. In *Proc. 13th European PVM/MPI Conference*, Bonn, Germany, September 2006. Springer.

[16] J. Labarta, S. Girona, V .Pillet, T. Cortes, and L. Gregoris. DiP : A parallel program development environment. In *Proc. of the 2th International Euro-Par Conference*, Lyon, France, August 1996. Springer.

[17] E. Laure et al. Programming The Grid with gLite. In *Computational Methods in Science and Technology*, pages 33–46. Scientific Publishers OWN, 2006.

[18] R. Menday. The Web Services Architecture and the UNICORE Gateway. In *Proceedings of the International Conference on Internet and Web Applications and Services (ICIW) 2006, Guadeloupe, French Caribbean*, 2006.

[19] T. Moses et al. *eXtensible Access Control Markup Language*. OASIS Standard, 2005.

[20] W. Nagel, M. Weber, H.-C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12(1):69–80, 1996.

[21] M. Riedel and D. Mallmann. Standardization Processes of the UNICORE Grid System. In *Proceedings of 1st Austrian Grid Symposium 2005, Schloss Hagenberg, Austria*, pages 191–203. Austrian Computer Society, 2005.

[22] A. Streit, D. Erwin, Th. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and Ph. Wieder. UNICORE - From Project Results to Production Grids. In L. Grandinetti, editor, *Grid Computing: The New Frontiers of High Performance Processing, Advances in Parallel Computing 14*, pages 357–376. Elsevier, 2005.

[23] F. Wolf. *Automatic Performance Analysis on Parallel Computers with SMP Nodes*. PhD thesis, RWTH Aachen, Forschungszentrum Jülich, February 2003. ISBN 3-00-010003-2.