Distributed Workflow Management for Large-Scale Grid Environments

Joerg Schneider, Barry Linnert, Lars-Olof Burchard
Technische Universitaet Berlin, Germany
{komm,linnert,baron}@cs.tu-berlin.de

Abstract

Workflow management in large-scale Grid environments is a very challenging task centralized management systems are not able to cover sufficiently. Therefore, we present our Workflow On-line Resource Management (WORM) architecture built on top of active network technology. The approach integrates a peer-to-peer like organized workflow management system with existing or newly built management systems for the resources building the Grid. In our approach, each workflow is represented by a mobile autonomous entity which uses the active network infrastructure to move through the Grid, which is represented by an active overlay network on top of existing network infrastructure. Thus, control of the workflow execution is handed over to the autonomous code without requiring a central system to be in charge of the computation and cope with reservation, failures, etc. The WORM architecture is presented together with a classification into the taxonomy of workflow management systems.

1. Introduction

In this paper, we present a novel architecture for workflow management in large-scale Grid environments. As the amount of workflow jobs increases within the Grid the importance of providing sufficient support for these workflows rises, too. The *Workflow Online Resource Management* (WORM) approach provides such features needed to build advanced support for workflows in next generation large-scale Grid environments.

As a typical example for a complex workflow in a Grid scenario, an application is depicted in Fig. 1. The workflow processed in the distributed environment consists of five sub-tasks which are executed one after another in order to produce the final result, in this case the visualization of the data. This includes network

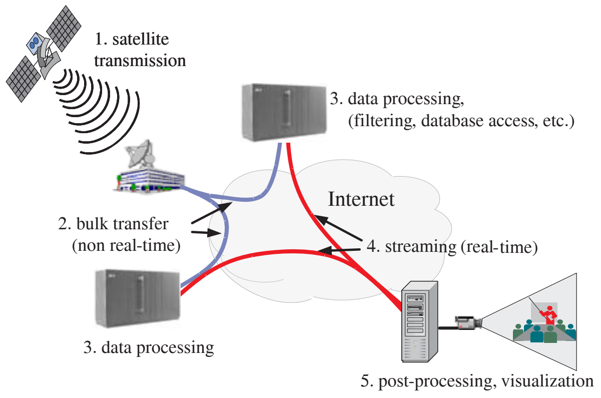
1. satellite transmission
3. data processing, (filtering, database access, etc.)
Internet
2. bulk transfer (non real-time)
4. streaming (real-time)
3. data processing
5. post-processing, visualization

Figure 1. Example of a complex workflow using the Grid

transmissions as well as parallel computations on two cluster computers.

Because, next generation Grid environments will consist of large amounts of different resources ranging from computing devices, such as parallel computers and network infrastructures, storage capacity, to human resources, centralized control instances can be hardly extended to support the large number of requests all the workflow jobs generate. So the complexity arising from extensive composed workflows consisting of requests for the distributed and diverse resources turns up to overstrain any centralized approaches [7]. Additionally the resources often belong to different domains and organizations this effect is increased. The WORM architecture is up to overcome the limitations of centralized approaches.

The resources building the Grid usually provide their own management environment. These environments range from operating systems to resource management systems for a couple of resources, such as Globus [8] or the VRM [1]. As these resources are distributed within the Grid the WORM architecture has to integrate theses management systems in a transparent and distributed manner.

To build a complex workflow different resources need to be allocated and released accurately. For this the WORM architecture provides a workflow control and execution engine dealing with these tasks. So the WORM itself can be seen as be situated on top of the local management systems of all of the Grid resources needed to run the workflow application. Using the WORM architecture the user is now able to pass the responsibility in workflow execution and control to the Grid itself, independent from any single and centralized execution entity.

To have a reliable and extensible framework on which the WORM architecture can be based on, the active network technology (ANTS) [14] was chosen. So the WORM approach can abstract from defining, implementing, and deploying a complete new infrastructure to access all the different distributed resources.

The workflow application itself can be specified using different models and approaches. The WORM architecture implies no limit to the expressiveness of the workflow specification.

So, implementing automated, adaptive, and flexible allocation of arbitrary resources in a distributed and coordinated manner the WORM architecture abandon any centralized logic.

After presenting former work related with Grid workflow management and Active Networks, the architecture of the WORM is described in Detail. In the following section we use a taxonomy for Grid workflow systems to classify the WORM. We conclude the paper with an outlook on the future development of the WORM architecture.

2. Related Work

Connecting different resources on several locations to a Grid is widely researched. Most research work focuses on middleware to allow an unified access to these resources and to manage single jobs for only one resource. Several architectures for Grid resource management have been developed, e.g., the VRM architecture [1] and the Globus toolkit [8].

Based on these results the composition of complex workflows using services provided by Grid resources is currently researched. At present there is not as much work done as for simple Grid jobs.

To declare the composition of workflows a number of description languages have been proposed. Some of them are based on well known languages for behavior modeling, like petri nets, e.g., used in the Fraunhofer Resource Grid [9]. Another approach is used for the Grid service flow language (GSFL) by adopting concepts from the web service composition domain [10].

The GRAAP working group of the Global Grid Forum (GGF) works on a specification for modeling of service level agreements [3]. The specification does not provide any means to describe Grid workflows, but it introduces techniques to define multiple Grid jobs in a single agreement, which can be extended to a workflow description.

The architectures proposed for Grid workflow handling are usually composed of an user tool and the workflow execution engine. In order to specify the workflow, the user tool composes the sub-tasks. The user tool also calls the execution engine which controls the execution of the workflow within the Grid [16, 9, 2]. In some architectures there are additional layers to enhances the workflows, e.g., by splitting up abstract tasks into concrete sub-workflows [9].

The workflow execution engine is usually realized as a central instance that interacts with the Grid. In some cases the used Grid resources exchange the input and output data directly, but even then there is an additional central instance coordinating the control flow [10]. In the GridFlow architecture [2] multiple execution engines are used, but the control flow of a workflow is always handled by the same system.

In [7] the performance of such centralized execution engines has been compared with two models of distributed execution engines. As the simulations showed, the performance advantages of the decentralized models are significant. Unfortunately the work was only done using simulations, thus a specific architecture realizing this distributed execution is missing.

Yu and Buyya proposed a taxonomy to compare Grid workflow management systems [18]. In this paper this taxonomy is used to analyze how the WORM architecture could be classified using this taxonomy. This classification can be used to compare the WORM approach with other Grid workflow handler.

In [14] a introduction into the concept of active networks is given. In an active network the routers are enhanced such that they can execute user-given code in order to deploy new protocols easily. Even load balancing and error handling mechanisms can be installed in such a way. While the active network was designed to substitute the routing mechanisms, the nodes in the active network can also modify the content and target of bypassing packets. As it can be easily seen, such a flexibility requires strong security mechanisms, too.

The Janos Node OS [15] provides an extension for the router operating system as well as an execution environment for the active network code. As execution environment works an extended Java Virtual Machine.

The Janos system contains a number of security mechanisms, e.g., to limit the resource usage of a single application and to decide which application handles which packets. Other active networks frameworks are either bundled deeper with the operating system kernel, e.g., Silk [12], or as loose as Janos, e.g., AMP [6], which runs on top of various operating system.

3. WORM Architecture

Our approach relies heavy on the distributed realization of all the tasks of the resource management. In this way our WORM architecture differs from common Grid architectures and workflow management systems.

These workflow management systems often depend on centralized infrastructures with a single management entity. The centralization of most of the tasks, the management has to cover, leads to various disadvantages. Especially in large-scale Grid environments the range of different workflows requesting huge amounts of various resources can generate excessive load which will lead to unacceptable response time of the central unit or even the overload of the central unit itself. In cases of failures of the central entity all of the workflow jobs handled will be affected and in the worst case lost.

An alternative to these centralized architectures is to use an architectural framework such as WORM which is described in this paper. The architecture completely distributes the management task, describing workflows as mobile *worm* code where the job control follows the workflow execution.

The Grid workflow execution environment consists of the following components: Grid resources, *infrastructure nodes* (IN), which serve as *gateways* to the resources, and the underlying active network infrastructure. The actual worm is represented by code entities (*WORM instance*), running on the INs of the workflow environment and moving across the resource network.

3.1. Workflow Modeling

As described before, a workflow management system is usually divided into two parts. The execution engine and the user tools to define the workflow and to submit it to the execution engine. In this paper the focus lies only on the execution engine. For the specification of the workflow we suggest to use existing tools. Actually most of the existing tools can be easily adopted to cooperate with the WORM architecture. Nonetheless, some properties of the possible workflows have to be defined.

In our framework the resource management, i.e., the WORM instance, decides where a subjob of the workflow will be executed. Therefore, there is no need to specify the machine in the workflow. The workflow can be also an acyclic graph, as long as an adequate mechanism to model conditions is implemented.

This conditions must be decidable using the return code of the sub jobs. This limitation is needed, as the worm will run on the machines dedicated to resource management and so wont be able to do complex analysis of the output data by itself. However, this analysis can be defined as a separate sub-job such that it will be done by dedicated compute resource and will not disturb the resource management.

As the WORM framework does not only manage compute resources, but also network bandwidth, the data that has to be transfered from a preceding subjob to the following one must also be specified. Based on this information the required bandwidth will be allocated exclusively for this transmission if the network supports such allocations.

3.2. WORM instances

WORM instances are pieces of code which define the workflow execution process. Users encode the workflow description, producing restricted Java code using the interfaces of the IN.

As the code is executed in the restricted environment of an IN the possible harm of user given code is reduced. At the same time executable code serves as a powerful workflow description language used by the workflow execution engine.

There is no need to specify the workflow manually as the code is more used as the internal representation within the workflow execution environment. The build process of this code may be supported by tools. So, a workflow description, e.g. modeled in GSFL [11], can be easily translated into worm code.

3.3. Infrastructure Nodes

The general architecture of an IN within the WORM architecture is outlined in Fig. 2. The foundation of an IN is an active network node, in our case using the Janos Node OS [15] in conjunction with the Java-based ANTS toolkit [17], upon which the workflow execution environment is situated. Each IN is based on an active network node, i.e., a node in a network which provides an execution environment for mobile user-defined code. Active network nodes already provide the needed functionalities to receive, execute, and migrate code. In

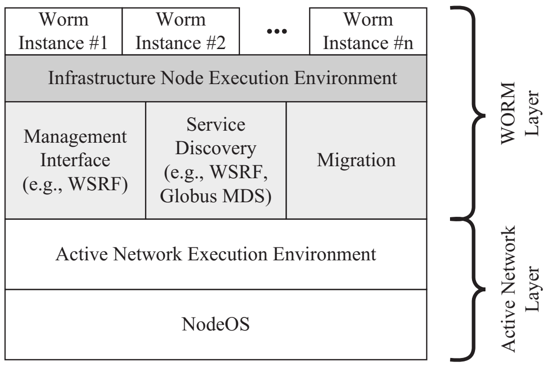
Worm Instance #1
Worm Instance #2
...
Worm Instance #n
Infrastructure Node Execution Environment
Management Interface (e.g., WSRF)
Service Discovery (e.g., WSRF, Globus MDS)
Migration
Active Network Execution Environment
NodeOS
WORM Layer
Active Network Layer

Figure 2. Architecture of an infrastructure node (IN).

common active network environments, this code represent network packets, in our framework the code represent WORM instances. In addition, each IN exposes interfaces required to process and control the actual workflow jobs. This includes management functionality, i.e., for allocation and execution of processes.

In order to facilitate the basic functionality for a Grid workflow infrastructure, the INs must provide the following services.

Service Discovery This is essential to identify available resources and their locations in the network including information such as the IP address of the resources and their type, e.g., cluster computers. The whole service discovery infrastructure can be implemented using peer-to-peer organization mechanisms [13] or basing on standard Grid middleware such as Globus MDS [5].

Management Interface The management interface of a local resource needs to provide the required functionality to allocate and access the requested resources, and to perform the workflow job encoded in the WORM instance. The interface also provides means for controlling a job running on the Grid resource. This can be implemented with standardized protocols [4] by either using the resource's management interface directly as described in [1] or using the management interface of the Grid middleware.

Migration Throughout the workflow processing, the WORM instances – including the current status of the workflow – must be migrated together with the other workflow data, e.g., program code and input data or results. In the active network setting, this corresponds with routing a packet and therefore, is already implemented in the active network infrastructure.

WORM instances determine each workflow sub-job, search for suitable resources (service discovery), select and allocate resources (management interface), and migrate to the resource together with the workflow data and programs. Each WORM instance performs all of the management tasks completely on its own, e.g., using WSRF [4], and therefore contains the current status of the workflow.

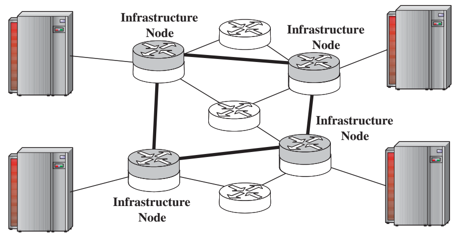

Figure 3. WORM infrastructure: the active network nodes (gray) built an overlay network.

The advantages of using active network nodes for the IN execution environment are many-fold. For instance, active network architectures already provide a safe and reliable execution environment for mobile user-written code, preventing malicious code from harming the infrastructure and avoiding that too many resources are consumed [15]. A network supporting bandwidth management already needs such complex management functionalities as they are possible using active networks. Therefore we assume, that a managed network already incorporates active network technology or it can be easily deployed there. Thus, it is not required to implement a completely new infrastructure. In the notion of active networks, the workflow code is defined as a new network protocol and WORM instances are treated as executable packets. The active network infrastructure builds an overlay network (see Fig. 3), connecting the entirety of Grid resources.

In Fig. 4, the setting on a single infrastructure node is outlined. The active network nodes serve as *gateways* to the actual resources, i.e., WORM instance execution and workflow job execution are decoupled. Job execution is remotely controlled by the WORM instance. When the resource in question is network bandwidth, the IN serves as gateway to the remaining network infrastructure. In order to participate in an existing Grid, a new Grid resource just has to set up a new active network node on a gateway and publish its service.

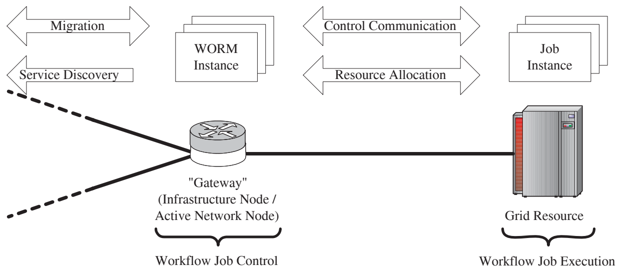

Figure 4. Grid resource with gateway.

The gateway is not necessarily always a router, but can also be set up on the Grid resource itself, e.g., a dedicated cluster node. An elegant feature of active networks is that missing code, e.g., the management functionality, is downloaded to the new node. In addition, functionality upgrades for INs can easily be distributed.

4. Classification

As described before, Yu and Buyya proposed in [18] a taxonomy to compare Grid workflow management systems. In this section this taxonomy is used to classify the WORM architecture presented in this paper.

4.1. Workflow Design

The taxonomy starts with the analysis of the workflow design. As the architecture covers the workflow execution only, no concrete workflow design model is given. But the requirements for possible workflow design tools and workflow languages will be presented.

The WORM architecture is designed to support Non-DAG structures in the workflow. But in the current implementation a feature of both DAG as well as Non-DAG structure is not fully supported: Parallelism. While it is possible for a single WORM instance to control more than one resource at once, it is not possible to split the WORM instance. To implement the idea of the control flow instances following the workflow execution fully the splitting of worms into independent ”sub-worms” would be required.

The WORM instance is searching for matching resources for each step of the workflow, thus it is possible to submit an abstract workflow model. In each step the concrete execution environment for this step will selected by the WORM instance.

The workflow composition system is not part of our architecture. If we would take into account the input data needed to start a WORM instance, a user driven, language based composition system would be used. In the current state restricted Java code written by the user is used as input, but in future a composition system using a graph-based modeling language will support the user to generate this input.

Quality-of-Service aspects are also part of future work, as it is not yet clear how to ensure the service level with a control instance running on various systems maintained by different organizations.

4.2. Information Retrieval

The retrieval of information will be performed on demand. Only so called dynamic information is used. The information will be provided either by already deployed Grid information systems or by other infrastructure nodes on a peer-to-peer base.

4.3. Workflow Scheduling

The main feature of the proposed architecture is the distribution of the workflow execution engine. Therefore the scheduling is done distributed as well. The taxonomy describes a distributed scheduling architecture as a system of individual schedulers forwarding jobs to each other in case of overload. In contrast, in the proposed architecture the scheduler takes into account resources in other domains every time. Thus, it acts in some kind like a centralized scheduler with global knowledge, but it is a distributed approach as there is a scheduling instance in each infrastructure node and the scheduling of the tasks of the workflow are done by a number of scheduler and not always by the same one.

Based on the current architecture the WORM instance just searches for a resource to execute the next step within the workflow. The WORM instance performs only local decisions according to the notion of the taxonomy. As the workflow model, which a WORM instance is based on, is abstract the architecture uses a dynamic, just-in-time planning.

4.4. Intermediate Data Movement

The concept of the WORM architecture was developed based on the idea of letting the data flow following the workflow. In the proposed architecture the input and output data is moved on a peer-to-peer base from resource to resource. But even the user given configuration data moves in the same way the workflow is executed, as it is part of the WORM instance.

In the notion of the taxonomy this means the data movement is automatic and peer-to-peer based. Automatic data movement stands for start and control of the data transfer is done by the workflow execution engine.

5. Conclusion and Future Work

The WORM framework is designed to realize all tasks of the workflow management in a distributed manner. The instance responsible to control the allocation of resources, the transmission of needed data and the execution of the sub-jobs is always situated near the executing machine. Former simulations showed the superior performance of such workflow execution engines, while we present a concrete architecture to implement the engine in such a way here. The presented framework can be easily deployed in a network already using active network technology. In other scenarios each computer can be turned into an active network node and thus into a WORM infrastructure node. Using the active network technology and the Janos Node OS many established security and communication mechanisms can be utilized. To make the WORM architecture comparable with other workflow management systems we used the taxonomy of Yu and Buyya to classify our framework.

Future work will deal with the problem of executing parallel parts of the workflow by splitting worms into several instances and coordinating these instances. Besides this issue, an interesting question is how quality-of-service guarantees may be integrated into our framework. In the current setup, only the allocation and reservation of a single resource at a time is supported which can be extended to cover guarantees for several consecutive resources. This requires additional functionality, such as distributed deadlock detection and remote allocation. It may be unrealistic to provide guarantees for complete workflows, e.g., deadlines, since this demands for allocation of any required resource in advance which obstructs the distributed character and self-organizing features of the WORM architecture. However, less stringent constraints may be satisfied, e.g., owners of worms might be notified of allocations for a few stages in advance. This means, allocations are made more than one step ahead.

References

[1] Burchard, L.-O., M. Hovestadt, O. Kao, A. Keller, and B. Linnert. The Virtual Resource Manager: An Architecture for SLA-aware Resource Management. In *4th Intl. IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGrid), Chicago, USA*, 2004.

[2] Cao, J., S. Jarvis, S. Saini, and G. Nudd. GridFlow: Workflow Management for Grid Computing. In *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), Tokyo, Japan*, pages 198–205, May 2003.

[3] Czajkowski, K., A. Dan, J. Rofrano, S. Tuecke, and M. Xu. Agreement-based Service Management (WS-Agreement, Draft). http://www.gridforum.org/Meetings/GGF11/ Documents/draft-ggf-graap-agreement.pdf, 2004.

[4] Czajkowski, K., D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. The WS-Resource Framework. http://www.globus.org/wsrf/specs/ws-wsrf.pdf, 2004.

[5] Czajkowski, K., S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*, page 181, Washington, DC, USA, 2001. IEEE Computer Society.

[6] H. Dandekar and A. Purtell. Amp: Experiences with building an exokernel-based platform for active networking. In *DANCE '02: Proceedings of the 2002 DARPA Active Networks Conference and Exposition*, page 77, Washington, DC, USA, 2002. IEEE Computer Society.

[7] Feng, Y., W. Cai, and J. Cao. A Simulation Study of Job Workflow Execution Models over the Grid. In *Grid and Cooperative Computing: Second International Workshop, GCC 2003, Shanghai, China*, volume 3033 of *Lecture Notes in Computer Science (LNCS)*, pages 935–943. Springer, December 2003.

[8] Foster, I., C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *7th International Workshop on Quality of Service (IWQoS), London, UK*, pages 27–36, 1999.

[9] Hoheisel, A. User Tools and Languages for Graph-based Grid Workflows. In *Workflow in Grid Systems Workshop in GGF10 at Global Grid Forum*, 2004.

[10] Krishnan, S, P. Wagstrom, and G. von Laszewski. GSFL: A Workflow Framework for Grid Services. Technical Report Preprint ANL/MCS-P980-0802, Argonne National Laboratory, Aug 2002.

[11] Krishnan, S., P. Wagstrom, and G. von Laszewski. GSFL: A Workflow Framework for Grid Services. Preprint ANL/MCS-P980-0802, 2002.

[12] N. Shalaby, Y. Gottlieb, M. Wawrzoniak, and L. L. Peterson. Snow on silk: A nodeos in the linux kernel. In *IWAN*, pages 1–19, 2002.

[13] Talia, D. and P. Trunfio. A P2P Grid Services-Based Protocol: Design and Evaluation. In *10th International Euro-Par Conference, Pisa, Italy*, volume 3149 of *Lecture Notes in Computer Science (LNCS)*, pages 1022–1031. Springer, September 2004.

[14] Tennenhouse, D. and D. Wetherall. Towards an Active Network Architecture. *ACM SIGCOMM Computer Communications Review*, 26(2):5–17, April 1996.

[15] Tullmann, P., M. Hibler, and J. Lepreau. Janos: a Java-oriented OS for Active Network Nodes. *IEEE Journal on Selected Areas in Communications*, 19(3):501–510, March 2001.

[16] von Laszewski, G., K. Amin, M. Hategan, and N. J. Zaluzec. GridAnt: A Client-Controllable Grid Workflow System. In *37th HawaiŠi International Conference on System Science*, 5-8 Jan. 2004.

[17] Wetherall, D., J. Guttag, and D. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *OPENARCH 98*, pages 117–129. IEEE, April 1998.

[18] Yu, J and R. Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. Technical Report GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, March 10 2005.