

IST-2001-32133

GridLab - A Grid Application Toolkit and Testbed

Use Cases for Adaptive Components

Author(s):	Thilo Kielmann, Gabrielle Allen, Matthew Shields, Ian Taylor, André Merzky, Jarek Nabrzyski
Document Filename:	GridLab-7-UCR-0001-UseCasesReport
Work package:	WP7: Adaptive Grid Components
Partner(s):	VU, AEI (for WP1 and WP2), UWC (for WP3), ZIB (for WP8), PSNC (for WP9)
Lead Partner:	Vrije Universiteit (VU)
Config ID:	GridLab-7-UCR-0001-1.0
Document classification:	IST

Abstract: This report describes a set of use cases for adaptive components to be addressed in the course of the GridLab project. The use cases have been determined by the needs of the partner work packages mentioned above.



Contents

1	Introduction	2
2	Relation to other work packages	2
2.1	Relation between WP7 and WP9	3
3	Use Cases	3
3.1	Throughput Optimization of Triana Flow Graphs (1)	3
3.2	Throughput Optimization of Triana Flow Graphs (2)	4
3.3	Parameter Adaptation for Parallel I/O	4
3.4	N-to-M Data Transfer Time Estimation	5
3.5	Remote Data Visualization	6
3.6	Communication Adaptation of Distributed Simulations (1)	6
3.7	Communication Adaptation of Distributed Simulations (2)	7
3.8	Distributing Computation across Heterogeneous Processors	7
3.9	Compute Time Estimation	8

1 Introduction

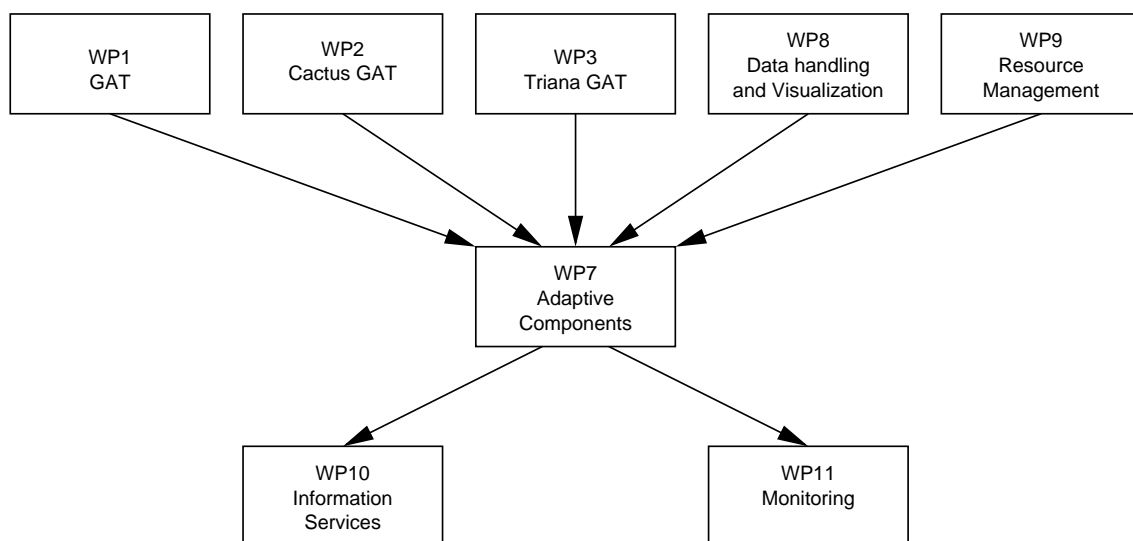
Grid resources are highly dynamic: network connections have varying bandwidth and latency, available CPU resources come and go. Static (inflexible) strategies and implementations are thus not suitable in Grid environments. Adapting application behavior (together with middleware layers) thus is inevitable.

In WP7, we will generalize the basic techniques of gathering relevant monitoring data, correlating this data with performance models (specific to toolkit modules), and short-term prediction of application performance, finally leading to behavior adaptation. The WP will develop generic components for implementing adaptive behavior.

The goal of this work package is to provide adaptive application components to other modules of the application toolkit. Common to these components is that they take dynamic information about Grid resources (network status, CPU availability, memory footprint, etc.) as input to module-specific performance-prediction models and implement adaptive strategies that let applications efficiently use the given resources. For grid-aware applications, this general mechanism is used for numerous purposes like data access, task scheduling, and dynamic application reconfiguration. Although use-case specific APIs and event models are necessary, the basic techniques are similar. The objective is to develop adaptation components that will be instantiated to their use from other modules of the GAT. For the input data, this WP will rely on the work performed in WP10 and WP11.

2 Relation to other work packages

The following simple diagram depicts the relation of WP7 with other work packages in GridLab. The diagram shows a is-client-of relation. Work packages 1, 2, 3, 8, and 9 thus are clients of WP7; adaptation components will be built for these work packages' needs. We call these the *client work packages* of WP7. WP7 is a client of work packages 10 and 11. Work package 11 provides the infrastructure for providing actual monitoring data while WP10 provides meta data about the monitoring system like contact information and metrics descriptions.



Technically speaking, the adaptation components will become plugins to the GAT modules of the client WPs of WP7. Other parts of the GAT, or even applications, will not observe their presence directly.

2.1 Relation between WP7 and WP9

WP7 supports WP9 by providing adaptation components for the purposes of the resource management system built by WP9. Besides that, both work packages complement each other in their functionality for the overall project goals. This can be summarized as follows:

- WP9 is responsible for resource management in general. As such, it builds a resource broker and an application scheduler, forming substantial parts of an overall *GridLab Resource Management System* (GRMS). All allocations (assignments, scheduling) of grid resources to applications are supposed to be handled by the GRMS.
- WP7 is responsible for application-level optimization in order to efficiently exploit resources by an application, after the resources have been allocated by the GRMS. This may involve placement and scheduling of application subtasks within the GRMS-allocated resources.
- An adaptation component (as built by WP7) may indicate the necessity for additional resources during an application run. In this case, such additional resources will be requested via the GRMS. Application migration is one example of such resource allocation at runtime.

3 Use Cases

In the following, we list the individual use cases for adaptation components to be addressed throughout the GridLab project. The primary selection criterion for the use cases is the need for adaptive behavior from the client work packages. Of secondary interest is to achieve a coverage of possible resources (like network bandwidth, CPU speed, memory, I/O throughput), the changing availability of which causes the need for adaptation.

For each use case, we describe the scenario and what can be gained by adaptation to certain resources. We list the respective client work packages and the resources to be dealt with. We also outline the basic functionality of an adaptation component for the use case. However, we do not fix the details of an API between the adaptation component and its clients yet. Only after completion of GridLab's general architecture and service provision mechanisms, those APIs can be developed.

3.1 Throughput Optimization of Triana Flow Graphs (1)

Scenario: Triana programs are formed by so-called *units*, connected to a flow graph, through which data items flow from sources to sinks. Each unit performs some computation to process the data. Units may have multiple inputs and outputs, and they may be composed from other (simpler) units.

In a Grid environment, a Triana flow graph is supposed to run on a distributed set of machines. The units need to be mapped onto those machines for execution. The problem is to assign the units to the machines such that the overall system computes as fast as possible (with maximal throughput). Care needs to be taken of machines having different computing speeds and units having different computational needs.

Triana needs to estimate the runtime of given (compound) units on the given machines. Having those estimates, the overall throughput of a Triana flow graph can be optimized. In this (simplified) scenario, we assume the Triana graph to be computation-bound, so network-related aspects like getting data fast enough from one machine to another can be ignored safely.

Client WPs: WP3 (Triana GAT).

Resources: CPU speed of individual machines.

Basic Functionality: The estimation of unit execution time can be based on empirical measurements. This may happen during an initial benchmarking phase (at program startup). Ideally, all units of a program's flow graph are run simultaneously on all machines with small data sizes. One open research problem is to determine a small and feasible number of different data sizes with sufficient expressiveness for estimation of a unit's (non-linear) computational complexity. Based on these benchmark results, we expect to estimate the throughput of each unit on each machine with the data size of the real application problem.

Benchmarking the actual units on the actual machines gives the best estimation, as it includes all peculiarities of the machines and their JVMs (Java virtual machines). If benchmarking the actual units becomes impossible in some cases, then a generic Java benchmark might be used instead (e.g., SciMark). However, loss of precision and expressiveness has to be expected in this case. As far as available, the units' annotations for expressing computational complexity in terms of the $O(n)$ notation can be used to support execution time estimations.

The adaptation component should be implemented as a Triana unit that simply gets called, benchmarks the individual units, and returns runtime estimation data. A second plugin Triana unit can then compute the optimal assignment of units to machines. Whenever the mapping of units to machines is supposed to change, this second unit can be called again to compute a new mapping. Such mapping changes result from changes in the application's problem size, or from the availability of additional machines, or the removal of machines from a running application.

3.2 Throughput Optimization of Triana Flow Graphs (2)

Scenario: Extend the previous scenario with network-related aspects. Now, also take the available network bandwidth between Triana units (on different machines) into account. This scenario assumes that large datasets are sent between the individual units.

Client WPs: WP3 (Triana GAT).

Resources: CPU speed of individual machines, network bandwidth.

Basic Functionality: The optimization problem combines the throughput of both units and networks. The required network throughput can be derived from the time a unit needs to process a set of input data, and the size of this input data. Data sizes can be derived from the data types being processed by Triana units.

The computing times of the Triana units can be estimated as in the previous scenario. Network bandwidth is taken from the monitoring infrastructure. The adaptation component takes time series of available network bandwidth and predicts the short-term future of the network. Correlation of the raw network parameters with data transfers between Triana units produces the parameters for the unit-mapping optimization component. This component now also needs to be called in the case of significant changes of the underlying network parameters.

3.3 Parameter Adaptation for Parallel I/O

Scenario: A parallel application running on a single, parallel machine, performing I/O to locally attached disks. The choice of filesystem (local per CPU or remote via a LAN network) and the type of I/O layer strongly influence application performance. This has to be selected

individually for each parallel machine that has actually been allocated to an application. Additionally, the CPU stride (the selection of CPUs that actually perform I/O) for parallel I/O needs to be taken into account.

This adaptation can be performed either more generally via a GAT interface, or specific to Cactus (as part of CGAT). In the latter case, the term *application* should be replaced by *Cactus run*, and *I/O layer* by *I/O thorn*.

Client WPs: WP1 (GAT), WP2 (Cactus GAT), WP8 (Data Management).

Resources: I/O bandwidth between CPUs and local disks.

Basic Functionality: The choice between file systems and I/O layers can be made based on a benchmark inside the application run. This benchmark queries which I/O layers and file systems are available; and benchmarks them. (The benchmarked property is I/O bandwidth.) The benchmark is run by a single application process, with the help of a small group of peer processes for determining the I/O stride. This happens at program start and re-start after a migration. Benchmark results are stored in the adaptation component. Besides storing, the benchmark returns a choice of I/O layer and CPU stride.

During application runs, when reorganization becomes necessary (more CPUs, additional I/O turned on, changes of the computation), then the adaptation component can be re-queried for a new choice of I/O layer and CPU stride, without re-benchmarking (the hardware remains the same). The adaptation component is passive; it is called at application startup. In the case of application reorganization, it is called by the process that triggers the reorganization.

The choice of I/O layer and parameters, and the CPU stride is communicated to the parallel application. Cactus provides the steerable parameters (via the HTTP thorn) for this purpose.

3.4 N-to-M Data Transfer Time Estimation

Scenario: Migration of a parallel (e.g., Cactus) run from a machine with N CPUs to another machine with M CPUs. The time needed to transfer the application data between the machines needs to be estimated in advance, for possibly multiple purposes. First, a migration target machine may be selected by the time it takes to migrate to it (in behalf of the GRMS). Second, after a migration target has been chosen, one of possibly many network paths (e.g., testbed networks) may be selected. Third, after target machine and network has been selected, the transfer software/protocol may be selected (like GridFTP vs. scp). Finally, an estimation of data transfer time may be necessary to start migration in time, before the allocated compute time on the migration source machine ends.

Client WPs: WP2 (Cactus GAT), WP8 (Data Management), WP9 (Resource Management).

Resources: network bandwidth.

Basic Functionality: The monitoring system provides bandwidth estimates and network topology information. The bandwidth estimates are correlated to the throughput achieved by the available transfer protocols. Transfer time estimations can be computed from this. From multiple alternatives, the fastest one can be chosen. As *alternatives*, the choice might be between multiple migration target machines, between networks, between transfer software, and even between application-level strategies of migrating from N to M nodes.

The migration from N to M nodes involves the redistribution of application data between the compute nodes. This necessary redistribution can happen before, during, or after the transfer over the network, allowing to minimize transfer time by choosing the strategy that makes the most efficient use of the network.

An adaptation component can be passive and be called from the process that triggers the migration. The component returns a choice between the possible alternatives, depending on which level (from target machine to application-redistribution strategy) the selection is to be made.

In the case of estimating transfer time for starting migration in advance (before the compute time on the current machine ends), the adaptation component needs to be called from a separate watchdog process that monitors the progress of the parallel job, including compute time, available memory and disk space. Still, the estimating adaptation component remains passive.

3.5 Remote Data Visualization

Scenario: A user wants to visualize the output of a running, remote computation (like a Cactus run). The critical resource is the network bandwidth that limits either the quality or the delay of the visualization, or even both.

The user needs to determine a desired tradeoff between quality and delay. For this purpose, the user specifies the maximal useful resolution of the image, and a desired frame rate. The remote visualization software can then provide the best possible images within the constraints of the user and the technical possibilities.

Client WPs: WP8 (Data Management).

Resources: network bandwidth.

Basic Functionality: The key to this adaptation problem is to perform visualization using *progressive resolution*. The visualization software can construct the image hierarchically, starting with a very low resolution, up to the maximum quality that can be achieved within the constraints of frame rate and network bandwidth. The resolution may even vary between different parts of an image.

The adaptation component receives network bandwidth data from the monitoring system, produces short-term predictions, and correlates this with transfer times of image data over the measured link. The adaptation component can then provide an estimate for the image data volume that can be sent within a given time limit. This estimate can be used by the visualization software to control the generated image resolution.

3.6 Communication Adaptation of Distributed Simulations (1)

Scenario: A parallel Cactus simulation runs distributed across multiple machines. The simulation data is distributed across the individual nodes, where at the machine boundaries the array borders are mirrored to the respective neighbor machines in so-called ghostzones. Per simulation timestep, one ghostzone is necessary for computing. By increasing the number of ghostzones per node, the number of computable timesteps (before the next, updated ghostzone has to be received) can be increased, at the expense of replicating the computation for the ghostzone data to both respective neighbors. This is a critical issue at the borders of two parallel machines that are connected over a Grid network.

Using a single ghostzone minimizes the amount of necessary computation but synchronizes the remote machines for each timestep, while both machines have to wait until the message from their neighbor arrives. This waiting time is dominated by the network latency. By increasing the number of ghostzones, the frequency of this expensive synchronization is reduced, at the expense of additional, local computation. The optimization problem is to find a tradeoff between additional computation and synchronization overhead that minimizes execution time.

Client WPs: WP2 (Cactus GAT).

Resources: network bandwidth, network latency, CPU speed.

Basic Functionality: The adaptation component needs to find the ghostzone size at which computation and data transfer take about the same time, thus minimizing the synchronization overhead between neighboring machines. For this purpose, data about network latency and bandwidth are consumed from the monitoring infrastructure in order to predict the transfer time for the ghostzone data of a given size. The computation time to iterate over the local data (including the ghostzones) can be measured locally.

The input data is received continuously by the adaptation component which estimates a suitable ghostzone size. Either at regular intervals (like every n timesteps) or when the network parameters change dramatically, the adaptation component requests a ghostzone adaptation.

3.7 Communication Adaptation of Distributed Simulations (2)

Scenario: Extend the previous scenario with data compression for the exchanged ghostzones. This adds the further complexity of variability of message sizes and of additional computation time for compression and decompression. Both additional factors complicate the optimization problem. However, the compressed messages may allow for more frequent synchronization and thus for smaller necessary ghostzones.

Client WPs: WP2 (Cactus GAT).

Resources: network bandwidth, network latency, CPU speed.

Basic Functionality: The adaptation component now also needs to keep track of the actual data sizes being sent and of the time needed for compression and decompression. This adds to the complexity of the estimate and may lead to decreasing estimation accuracy that may lead to suboptimal results.

3.8 Distributing Computation across Heterogeneous Processors

Scenario: A parallel Cactus simulation simultaneously runs on processors of different speed. The problem is to distribute the application data (and with it the computational load) such that the overall simulation runs as fast as possible, balancing the load across the processors.

The load distribution needs to be estimated at program start and possibly during the run if severe load imbalance is detected, due to imprecise initial estimation, or due to the application dynamically changing its behavior (different simulation, analysis, I/O, etc.).

Estimation of computation time is a hard problem, because many parameters of the application and of the hardware platforms have influence on it. Therefore, a mix of information needs to be

taken into account, like architectural properties (e.g., *peak flops*), historical data from previous runs of the same application, and Cactus-specific micro benchmarks.

Client WPs: WP2 (Cactus GAT).

Resources: CPU speed.

Basic Functionality: The adaptation component needs to take whichever computation time information is available in order to estimate suitable workload distributions. For example, computation time information for an application from a machine of kind *A* might be available, but not for machine *B*. However, a *peak flops* like measure of both machines may enable to estimate the computation time of the application on machine *B*.

This adaptation component needs to access historical data about the machines (like *peak flops*), about previous application runs, and about Cactus-specific micro benchmarks. This information needs to be found, preferably via some general mechanism for finding performance data (in combination with the monitoring and information services work packages).

3.9 Compute Time Estimation

Scenario: The GRMS is supposed to schedule an application request such that the application completes within a given deadline (if at all possible). An alternative formulation of this problem is to determine the necessary resources in order to comply to a given deadline.

For this purpose, the scheduler needs information about predicted wait time in processor queues and about completion time of an application when scheduled to a given machine, or a combination of multiple machines, the latter in case of co-allocation.

Client WPs: WP9 (Resource Management).

Resources: CPU speed, processor queue status.

Basic Functionality: In this scenario, the adaptation component focuses on performance prediction for given time series of processor queue waiting times and of historical data for application run times. The application performance model needs to predict application runtime for a given application, a given problem data classification, and given machine architecture.