FRAMEWORK FOR DATA-INTENSIVE APPLICATIONS OPTIMIZATIONIN LARGE-SCALE DISTRIBUTED SYSTEMS

Cătălin CÎRSTOIU[1], Nicolae ŢĂPUŞ[2]

Lucrarea de faţă promovează ideea că problema aplicaţiilor data-intensive în sistemele distribuite de mari dimensiuni trebuie abordată depăşind simpla idee de planificare a cererilor de executat, căutând suport la nivelul platformei. Astfel, se propune ideea schimbării dinamice a configuraţiei resurselor folosite, pe baza cererilor efective, într-o abordare coerentă şi consistentă la toate nivelele. Concret, aceasta înseamnă considerarea reţelei nu doar o resursă importantă, ci un element activ al sistemului, ce trebuie monitorizat, urmărit şi optimizat.

Plecând de la aceste idei, sunt prezentaţi algoritimii şi serviciile de optimizare implementate în cadrul platformei propuse. În final, este arătată eficacitatea abordărilor alese în mai multe medii reale, precum Gridul ALICE sau sistemul de videoconferinţă EVO.

In this work we argue that when dealing with data-intensive applications in large distributed systems it is important to go beyond the simple scheduling of application tasks that have to run, and to support at the framework level the process of dynamically changing the configuration of the underlying resources, based on actual requirements, in a coherent and consistent manner. In practice, this means considering the network not only an important resource, but also an active element, whose use has to be monitored, tracked and optimized.

We have designed and implemented an optimization framework, together with several algorithms and corresponding optimizer services. Finally, we prove the effectiveness of our approaches in several environments, real such as the ALICE Grid or the EVO videoconferencing system.

Keywords: distributed systems, data transfers scheduling, network topology

1. Introduction

The experiments in the field of High Energy Physics (HEP) have proven the necessity for large distributed systems. The largest collaborations as of today, such as ALICE, ATLAS, CMS and LHCb of the CERN's Large Hadron Collider (LHC) [1] program, encompass thousands of physicists from hundreds of

1 Project Assistant, European Center for Nuclear Research – CERN, Geneva, Switzerland; Assistant, Faculty of Automatics and Computer Science, University POLITEHNICA of Bucharest
2 Prof. Faculty of Automatics and Computer Science, University POLITEHNICA of Bucharest, ntapus@cs.pub.ro

institutions in more than 30 countries worldwide. They drive the construction of a Data Grid [2] capable of handling massive datasets on a global scale. Managing these massive amounts of data requires two fundamental components at the network layer, on which higher-level services can be built: (a) a reliable, secure, high-performance data transfer protocol for use in wide area environments; and (b) proper management of the data transfer requests, in regard to the scheduling and handling of the ongoing transfers.

Another area of interest in the field of distributed data-intensive applications is the delivery of multimedia information in distributed computing environments. Typically, this employs multicast technologies which can be supported on regular Internet through overlay networks. The solution that was adopted to solve the multicast problem was called the *tunneling* approach. The tunnels should be established dynamically in order to form the overlay network that is used by the application.

We can clearly see a strong connection at the network level between these two kinds of applications. An efficient approach for both this problems consists in considering the network as an active element and an important resource, similar to computing and storage components, whose use should be monitored, tracked and optimized in real time. Moreover, latest developments in the field of high-speed networking brought closer and closer to the production status the idea of hybrid networks, trying to combine conventional parts of the network (packet switched) with circuit-oriented segments. This way, the most demanding applications make use of dynamically constructed optical paths which ensure the desired high end-to-end throughput.

This idea is actively investigated by projects such as DRAC [3], DRAGON [4], or OSCARS [5]. However, their focus is only on dynamic network configuration for the purpose of bandwidth time-slots reservations. Projects that handle transfer requests, like FTS [6] or Stork [7] consider only the end-points when performing data transfers scheduling, ignoring the common network segments on the actual paths.

Employing direct light-paths also reduces the cost of the entire system, as the cost of optical switches is much lower for high-speed data transfers, than the full stack of components in a 10Gbps layer 3 Router.

We have developed a framework for performing and handling data transfers and bandwidth requests with capabilities for network provisioning, based on FDT (Fast Data Transfer) [8] and MonALISA (Monitoring Agents in a Large Integrated Services Architecture) [9]. FDT is an application for efficient data transfers over wide area networks, capable of transferring data at disk (storage) speed. MonALISA is a large scale distributed monitoring system, with support controlling remote services from high-level clients. It offers a pluggable interface for monitoring and controlling modules. The MonALISA clients can also be

extended to implement custom behavior based on the monitoring information received from the distributed services, such as controlling the network topology.

2. Framework design

The role of the envisioned system is twofold: (a) optimizing the selection of the used network connections and (b) the efficient management of data transfers and bandwidth requests running over both regular and dynamically configurable network paths. To achieve this, the decisions taken both at the task scheduling level and the network topology level, must be based on the available bandwidth, delay and other concurrent activities on each segment of the network.

Overview

The connectivity between sites can be assured by a set of possible paths with different characteristics in regard to bandwidth, delay, quality of service or cost. By default, if no administration privileges are available on the end hosts, routing to various destinations is determined by the default site network configuration. However, if end system's parameters, can be changed, alternate but predefined paths, can be used instead the default. When paths can be dynamically setup by sending commands to various network devices, the flexibility of the system can be increased by providing more alternative paths (Fig. 1).

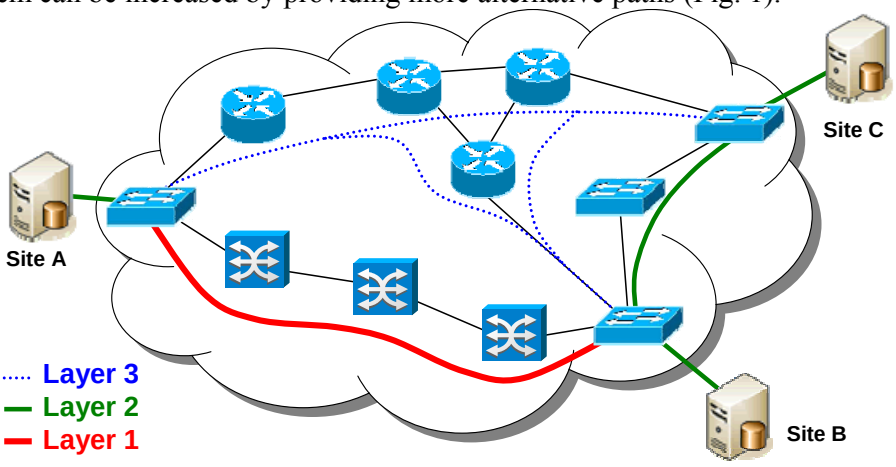

Fig. 1. Multiple paths in the network core available at different OSI layers

Setting up an alternative path is a process that could require activating each individual link on the path, by sending specific commands to the corresponding network device.

Having a path or virtual circuit setup, accomplishes the requirements related to quality of service. However, in case of massive data transfers, the

network and endpoint storages can become overwhelmed and fail to cope with all the traffic. This problem can be solved only through the *scheduling* of the incoming requests. Instead of performing each request independently, users will hand them to a *scheduling service* which will schedule their execution in such a way that the system doesn't become overloaded. The goal of this service is to make efficient use of the resources, minimize the response time and take care of errors by, for example, attempting to execute the requests several times, possibly on different paths.

Summing, in order to perform intelligent handling of the requests, an *optimizer* must know the topology of the network, its parameters and its current and future load.

The knowledge of the network topology is also vital for the multicast-based distributed applications. Optimizing the communication quality for this kind of applications requires continuous data feeds concerning the network, the nodes and the current traffic. Processing this data has to be fast and the decisions have to be propagated immediately from a *topology optimizer service* through the network so that the overlaid topology is kept within the optimum parameters.

We can distinguish here between two distinct, but closely related optimization problems: the *topology optimization* as a problem in its own and the *requests execution optimization*, which is an extension of the former, adding the time dimension.

Distributed services for monitoring and controlling network and tasks

The accomplishment of such a system can be achieved through a set of distributed services, loosely coupled and with well defined roles (Fig. 2).

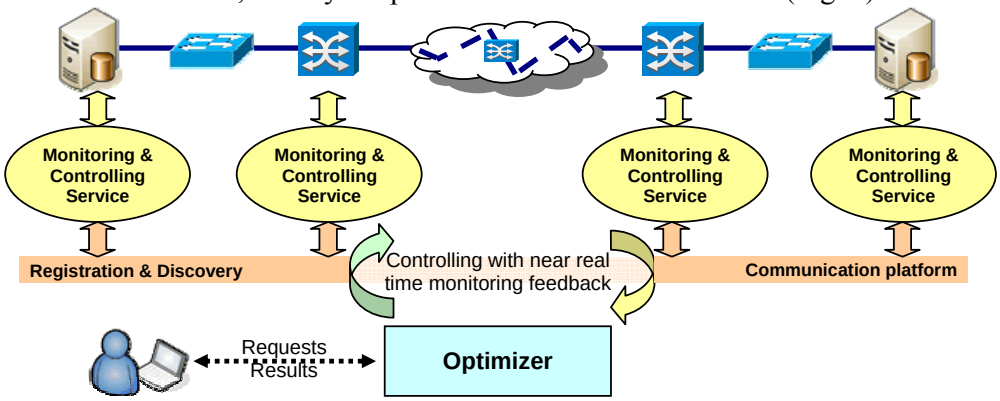

Fig. 2. Distributed services for monitoring and controlling end-to-end paths

The *monitoring and controlling services* are distributed throughout the system, on each end-point where requests are actually executed, or close to

intermediate points in the network that can provide monitoring information or that can be controlled.

The *optimizer* service coordinates the entire system based on the monitoring information received from the other services and on the users' requests. At the *optimizer*'s core lays an interface for pluggable *topology and scheduling algorithms*, based on the other components. This approach should allow for the evolution of the employed scheduling algorithms, as will be needed.

The interactions between the optimizer and the distributed services is assured by an advanced *communication platform*, which besides transporting monitoring and controlling messages, it provides registration and discovery.

In general, a *monitoring and controlling service* is responsible for discovering the local network topology and retrieve monitoring information about the local nodes and links. It makes this information available to the *optimizer* through the flow of monitoring data. The *optimizer*, by retrieving this information from all the existing services, can infer the entire topology.

A *monitoring and controlling service* that runs on an end-point should be able to setup the host in order to send the packets on the interface bound to the chosen path, optimize network related parameters (for example setup kernel network buffers size), start the transfer using some available protocol, monitor it and stop it. Finally, once the transfer is completed, it should again update the local configuration to the original settings. On an intermediary node, this service should monitor the network traffic as seen by the equipment (data provided by switches' and routers' counters), and it should be able to send commands to the network equipment or to other service that manages network paths to setup segments of the possible path.

Each network link advertised by the *monitoring and controlling service* can have associated procedures for setup or teardown, in controlling module's configuration. This way, the commands accepted by the distributed services can be generic, being particularized locally, on each service, based on the defined configuration. Thus, the underlying details of each administrative domain are hidden from the *optimizer,* making the whole framework flexible and extensible.

In this presentation, the optimizer is referred to as a single service, but this is only for the simplicity of the explanation. In reality, in order to assure the requirements for reliability and to avoid single points of failure, this service should be replicated and several similar entities should be available to users.

Abstracted network topology

The real network topology is abstracted as a graph within the *optimizer*, based on the available network segments reported by all the distributed monitoring services. This graph includes a set of nodes, interconnected through a

number of links with additional information. The algorithms implemented in the *optimizer* will run over this abstracted topology.

Some of these nodes represent real network devices or end-points. Likewise, some of the links are the representation of the actual underlying connections, others are only logical connections. However, due to the restrictions present in existent networks (i.e. firewalls), some of these nodes might not be visible to conventional network monitoring tools. Moreover, it could be impossible or unfeasible to add the complexity of the underlying network of various network domains. Therefore, we consider two types of network elements: real and virtual. For each of those, we can distinguish between manageable and fixed configuration network elements. An example of manageable virtual element would be another service that conFig.s a set of links on the computed path.

On the real network, we can distinguish between two types of paths: (a) *concrete network paths* – consists of the existing network links, which don't need any additional setup in order to be used, typically at layer 3 in the OSI hierarchy; and (b) *configurable paths* – end-to-end virtual circuits that already exist or that can be setup dynamically. Fig. 3 shows an environment supporting multiple path types, with the features and the commands suitable for each of them in the process of executing a request.

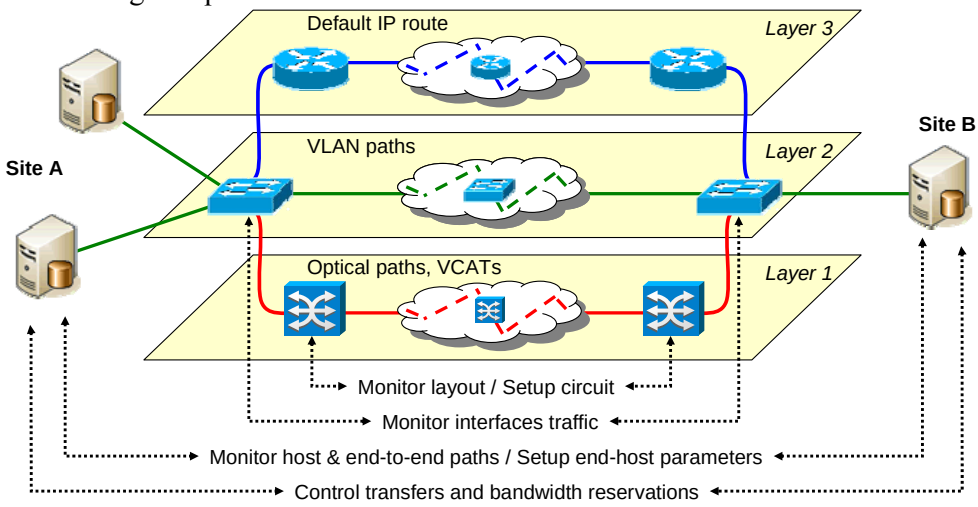

Fig. 3. Choosing and setting-up end-to-end paths at different network layers

In the abstracted topology, in order to support the paradigm of virtual circuits, with multiple links between two nodes and some of the links may be tagged with a certain path ID. Tagging is needed because not all possible paths in this graph can be followed in reality, due to actual routing configurations.

With this convention, describing the virtual circuits at levels 1 and 2 would mean adding links between the intermediary nodes, with a certain ID.

Supporting paths at layer 3 would mean tagging all these links with a default tag (for example -1). Now, finding a path from a source to a destination node is reduced to exploring this graph (based on a certain cost criteria) with the restriction that all the links in the path must have the same ID (Fig. 4).

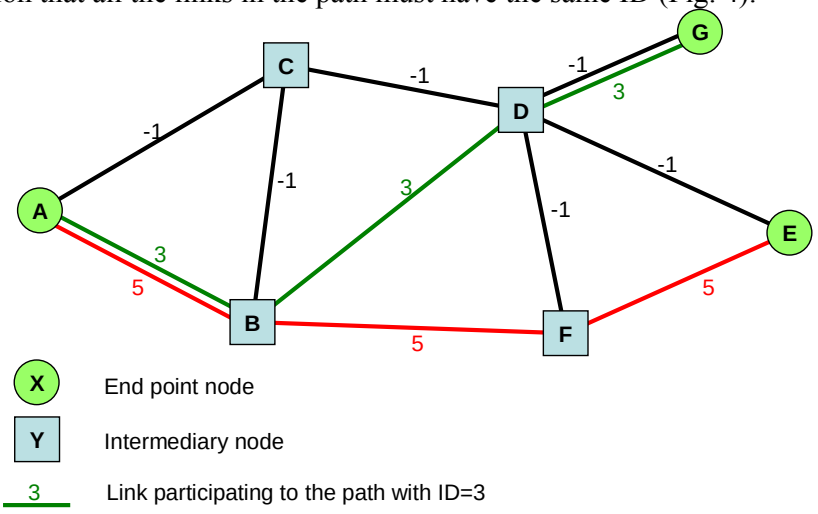

Fig. 4. Paths abstraction in the topology graph

Besides the regular monitoring information, each link in this graph has an associated queue with the current and future requests to be executed. This allows the other components in the optimizer to monitor the behavior of the network, to track the executing requests, and to decide how to execute the future requests.

3. Optimizing the execution of data transfers and bandwidth requests

The process of handling users' requests, either bandwidth reservations or data transfers can be seen as a two step procedure, requiring the solution for two problems: (a) the *path discovery problem* – finding possible paths between the two end-points involved in the request; (b) the *path allocation problem* – allocating the request to one of the available paths.

The *path discovery problem* (PDP) is a variant of the *breadth-first search* (BFS) algorithm [10] that we have developed for exploring graphs. The similitude comes from the constraint of finding shortest paths from source to destination. The difference of PDP from BFS stems from the possibility of having multiple edges between the same two nodes and their tagging. Another difference is the stopping condition: the algorithm will stop as soon as it finds the destination, instead of exploring the entire graph.

In the following two subsections we define and present the design details for two policies we have implemented for handling the path allocation problem:

(a) *time-based path scheduling* – the requests are scheduled in time over the existing links; (b) *dynamic bandwidth allocation* – priority based, immediate execution with dynamic bandwidth allocation for the existing requests.

Time-based path scheduling

The *path allocation problem (PAP)* concerns the allocation of all the network segments on the path associated with users' requests. The requests arrival in the system is *on-line*. Due to the fact that the capacity of the network is limited, the requests cannot be executed immediately, as they are received. System's goal is to execute all of them eventually, optimizing the resources usage. In our case, the allocated resources consist in the network segments, defined for the underlying physical network.

With a running system, when a new request is received, the scheduling process means finding the starting time for this request based on the estimated processing time (from the *prediction* module), the path (from the *PDP* algorithm) and the used bandwidth (as specified by user, in the path's limits).

There are two constraints, which differentiate this scheduling process from a classic job scheduling problem: (a) the request requires not a single, but a set of resources in order to be executed successfully – all the network link segments, are needed in order to perform a transfer, end-to-end; (b) a resource can be only partially used by a request – on the same network link segment several data transfers can be performed simultaneously, provided that the link's bandwidth is sufficient.

A visual representation of the constraints is depicted in Fig. 5. When scheduling a new request, we have to make sure that there is sufficient available bandwidth on all the link segments in the path, for at least an interval equal to the request's processing time.

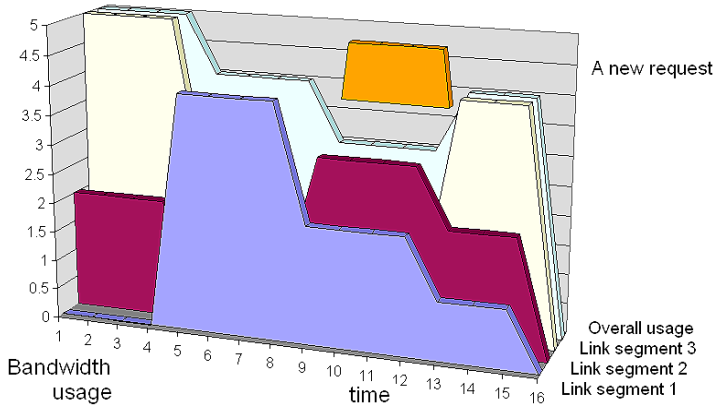

Fig. 5. Bandwidth usage over the links in a path

The complexity of the time-based path scheduling algorithms can vary from simple greedy approaches to complex algorithms and data structures for handling optimally all the needed details. However, in the current work we focus on the infrastructure itself, being concerned with its generality and flexibility to accommodate any scheduling algorithm.

Dynamic bandwidth allocation

The *time-based scheduling* policy works fine especially for batch-style requests. Regardless of the quality of the schedule and the efficiency of the scheduling algorithm, the handling of *high-priority requests* is not very good.

Typically, there are two basic approaches: (a) *non preemptive* scheduling, when the higher priority request is executed only after the end of the currently running requests (that affect the requested links) – which could mean an arbitrary long delay for the high priority request; (b) *preemptive* scheduling, when the currently running requests are interrupted and they are restarted (possibly from the beginning) when the higher priority request ends – which could mean the arbitrary delay of low-priority requests.

Another, novel, policy is to start the high-priority request immediately and reduce the bandwidth of the ongoing requests dynamically, without stopping them. This way, the two problems of classic time-based scheduling are solved. For this approach to work, it is needed that the underlying transfer protocol supports changing dynamically the bandwidth limitation.

The two policies are not exclusive and can be used simultaneously. In fact, in practice, dynamic bandwidth allocation should be supported by a time-scheduling mechanism, since the requests duration can vary in time, compared to the initially estimated interval.

4. Optimizing the overlay topology for a distributed application

A distributed application that uses an overlay network to communicate must take into account two main aspects in order to be able to perform efficiently: (1) *Routing* – since the routing of packets (at the *reflectors* level) is performed not by the underlying network, but at logical level, the application has to handle it; and (2) *Performance* – the *tunnels* that are selected to connect the reflectors have to be chosen in such a way that the links quality between all peers is maximized.

In order to support the first constraint, the typical choice is to build a *tree* over the possible connections in the mesh graph formed by all the participant *reflectors*. Using a tree for this purpose has two advantages: routing is simple – what is received by a reflector on a link will be forwarded automatically on all the other links; the amount of data that flows over the network is minim (Fig. 6).

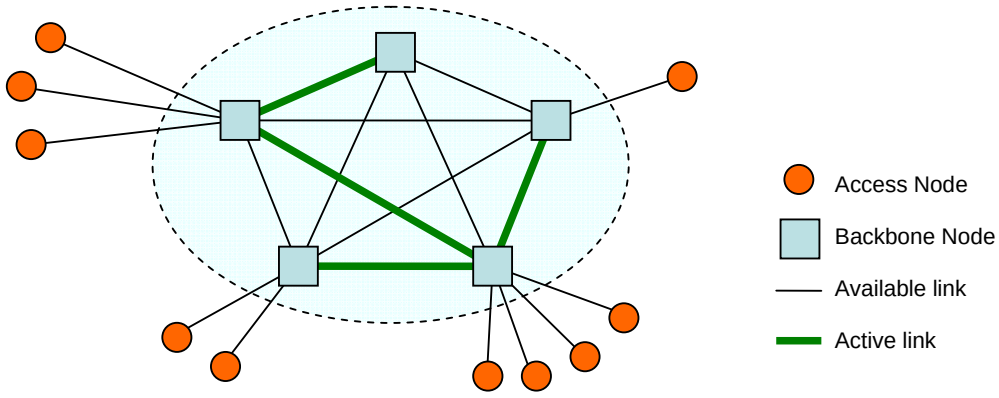

Fig. 6. Spanning tree over the actual network topology

Therefore, the tree topology must be enforced, to avoid cycles. If at some point cycles appear in this topology, packets will be forwarded continuously until the expiration of their time to live, which will generate high load on the reflectors and on the network. It becomes clear that the problem of optimizing the entire overlay topology for a distributed application involves finding the best *spanning tree* over the *graph* formed by the existing network segments.

The network topology optimization problem

The quality of this spanning tree is highly dependent on the quality of the links in the graph, which can be reduced to the *cost*, also known as *weight*, of the edges in the graph.

The *overlay topology optimization* (OTO) problem can be defined as follows: Given an undirected, possibly non-connex graph $G = (V, E)$ with real weights assigned to its edges, the problem consists in finding a set of non-overlapping minimum spanning trees $\{(V, T)\}$, one for each connex component of G, with $T \subseteq E$ with the minimal possible weight $w(T)$:

$$\text{OTO} = \{ e \mid e \in T, \text{ with } \min(\sum_{e \in E} w(e)), \text{ covering all } V \}$$

However, in real circumstances, this might not be sufficient for providing a reliable service for the distributed application that uses the overlay network. Before we can properly define the algorithm, we have to note that one must take into account a set of *time-dependent constraints* that apply on the spanning tree that is calculated. Thus, special care should be taken with new nodes that join the system at some point; or the nodes that join and leave in a quick sequence.

The idea of the *overlay topology optimization* (OTO) algorithm is to consider initially each node of the graph as a tree and then connect together pair of trees, until the whole graph is covered by a single tree. The process of joining is

performed gradually, choosing at each step the edge with the smallest weight that has its nodes in separate trees.

```
OTO(V, E)
      MST ← Ø
      NODE_IN_TREE ← Ø
      for each n in V do
            put(NODE_IN_TREE, n, n)
      made_connection ← true
      while made_connection do
            made_connection ← false
            best_e ← nil
            for each e in E do
                  if best_edge = nil or e.w < best_e.w then
                        t1 ← get(NODE_IN_TREE, e.n1)
                        t2 ← get(NODE_IN_TREE, e.n2)
                        if t1 ≠ t2 then
                              best_e ← e
            if best_e ≠ nil then
                  made_connection ← true
                  t1 = get(NODE_IN_TREE, best_e.n1)
                  t2 = get(NODE_IN_TREE, best_e.n2)
                  for each n in V do
                        if get(NODE_IN_TREE, n) = t2 then
                              put(NODE_IN_TREE, n, t1)
                  enqueue(MST, best_e)
      return MST
```

The overall complexity of the OTO algorithm is O(n·m).

5. Experimental results

To achieve our goals we have developed modules at all layers of the MonALISA framework. On the service side we have developed monitoring modules for connectivity (ping-like) and topology (traceroute-like); application control modules for links, bandwidth reservations and data transfers with FDT. On the client side we have developed two *Optimizer* services, one for each of the problems presented in the previous sections.

The next two subsections present the tests we have performed with the two optimizing services in order prove their efficacy.

Data transfers on the ALICE Grid

Since 2005, MonALISA is used in the process of monitoring the ALICE Grid [11], AliEn [12]. The data replication among AliEn storage elements is of particular interest for the ALICE community. In this section we present the results

of our tests with the *time-based path scheduling policy*, in the attempt to use the designed framework in the real environment of the AliEn Grid.

Regarding the network topology of the ALICE sites, we have built a logical configuration, which is shown in Fig. 7. This includes only the sites which have open ports in their firewall (needed for the actual data transfers with FDT), out of the 60 ALICE sites. If needed, this topology can be refined to include more details, as the local administrators would consider appropriate.

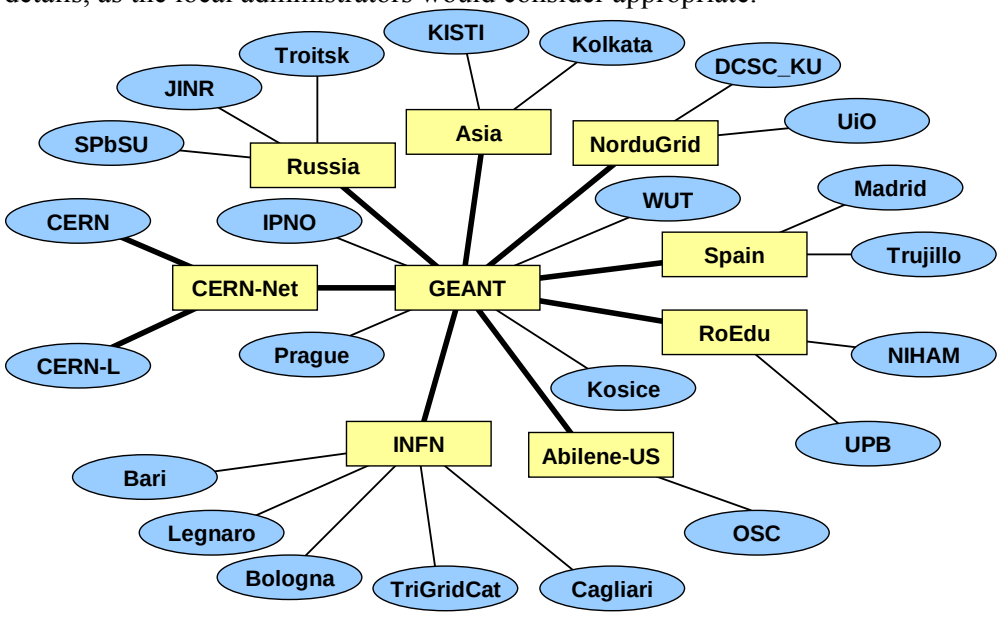

Fig. 7. Part of the ALICE sites and their assumed topology

For our testing purposes, the topology chosen for the participating sites was one of a tree, having as core nodes the major network providers and the sites as leafs. We have chosen the bandwidths of the links in such a way that from CERN to the others we have enough bandwidth for 10 x 100Mbps concurrent transfers, but the other networks support only 1 or 2 x 100Mbps simultaneous transfers. On this topology we have issued a set of transfer requests from both nodes at CERN, (CERN and CERN-L) to all the other participating sites. The *Optimizer* schedules the requests, using a time-based path scheduling policy, in three rounds, as in Fig. 8.

One can observe that there are no two concurrent transfer requests to two nodes belonging to the same group (regional network). The overlapping is allowed by the high capacity networks connecting the CERN nodes, but not by the other links.

Transfer Scheduler

Overview | Nodes | Links | Paths | Requests | Add Request | Add Batch Test | Admin Users | Log out (catac)

Nr.	RequestID	Status	From	To	Sched. Start	Estim. Duration	Duration	LimitBW	CrtBW
1-17	of 46	Any	Any	Any					
1	cat-439	SCHEDULED	CERN-L	UiO	26.02.2008 11:52	5 min		-	100 mbps -
2	cat-434	SCHEDULED	CERN-L	OSC	26.02.2008 11:48	5 min		-	100 mbps -
3	cat-438	STARTING	CERN	UiO	26.02.2008 11:47	5 min		-	100 mbps -
4	cat-437	STARTING	CERN	WUT	26.02.2008 11:47	5 min		-	100 mbps -
5	cat-436	RUNNING	CERN-L	Prague	26.02.2008 11:47	5 min		-	100 mbps -
6	cat-435	RUNNING	CERN	Kosice	26.02.2008 11:43	5 min		-	100 mbps 49.4 mbps
7	cat-433	RUNNING	CERN	OSC	26.02.2008 11:43	5 min		-	100 mbps 13.4 mbps
8	cat-432	RUNNING	CERN-L	Trujillo	26.02.2008 11:42	5 min		-	100 mbps 80.5 mbps
9	cat-429	RUNNING	CERN	Troitsk	26.02.2008 11:42	5 min		-	100 mbps 32.5 mbps
10	cat-428	RUNNING	CERN-L	SPbSU	26.02.2008 11:42	5 min		-	100 mbps 14.1 mbps
11	cat-426	RUNNING	CERN	TriGrid_Catania	26.02.2008 11:42	5 min		-	100 mbps 47 mbps
12	cat-425	RUNNING	CERN-L	Legnaro	26.02.2008 11:42	5 min		-	100 mbps 88.7 mbps
13	cat-424	RUNNING	CERN	Cagliari	26.02.2008 11:42	5 min		-	100 mbps 7.1 mbps
14	cat-431	FINISHED	CERN	Madrid	26.02.2008 11:37	5 min		5.3 min	100 mbps -
15	cat-427	FINISHED	CERN	IPNO	26.02.2008 11:37	5 min		5 min	100 mbps -
16	cat-423	FINISHED	CERN-L	Bologna	26.02.2008 11:37	5 min		5.3 min	100 mbps -
17	cat-422	FINISHED	CERN	Bari	26.02.2008 11:37	5 min		5.2 min	100 mbps -

Fig. 8. Scheduled, starting, running and finished transfer requests in ALICE

In Fig. 9 we present the actual transfer speeds, for the issued transfers. We can see the clear separation of the three rounds, conforming to the schedule.

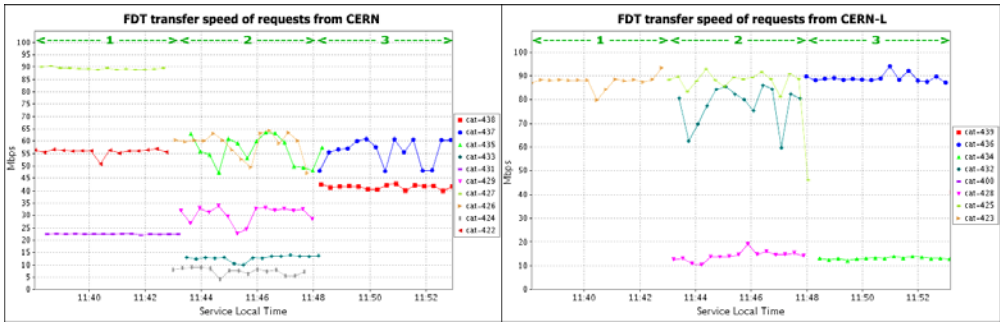

Fig. 9. Transfer speed of the requests from CERN and CERN-L to the other sites

With this example we have shown that scheduling of data transfers on a real Grid middleware is easily possible with our *Optimizer*. The data transfer speeds are relatively small in this case, since the transfers were performed between head nodes in each site, instead of the dedicated storage elements and paths, which are used in reality.

Optimizing the EVO topology

EVO [13] is a next generation videoconferencing system, based on a distributed architecture, which was created to provide support for the High Energy

Physics community. EVO evolved from VRVS [14], the focus being put on the capability of the infrastructure to adapt itself dynamically to the network status and ongoing changes, in order to ensure that the collaboration service runs without disruption.

We have adapted the *Global Connectivity Optimizer*, to run in the context of the EVO system. It receives monitoring information about the quality of the links (both available and selected in the current topology) and computes the minimum spanning tree that connects all the Panda servers (the reflector nodes).

The generated commands are sent through the communication platform and then they are forwarded to the Panda servers. These act on them, by changing the active communicating peers [15]. In Fig. 10 we show the generated tree.

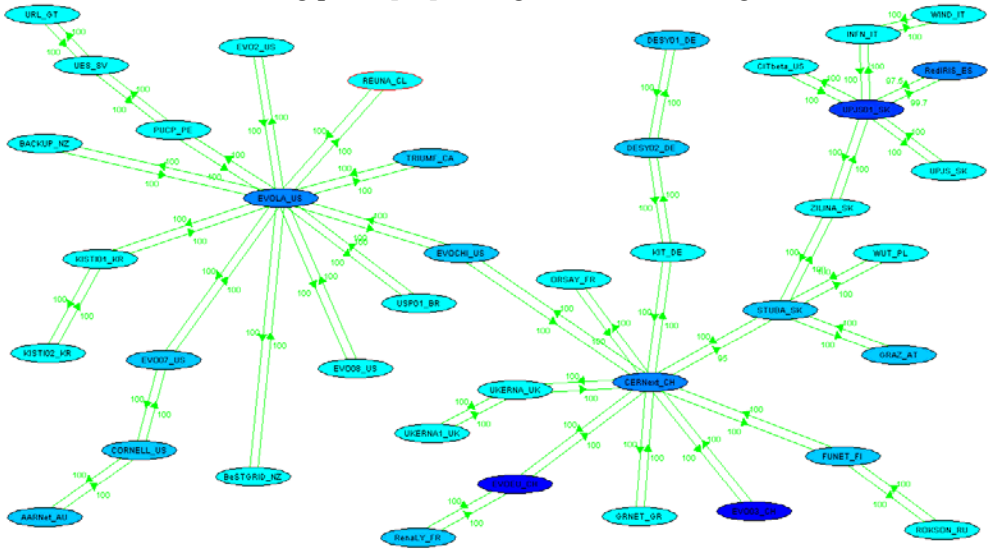

Fig. 10. Overlay topology - Minimum Spanning Tree

The cost of this tree is based on the monitored quality of the selected links. This quality varies in time according to the changes in the network. In general, if there are no major network events, it is rather stable. However, the situation is different if we look at system's behavior for a longer period of time, as shown in Fig. 11. The high peaks are generated by the penalized nodes. When a node is unstable, all its connections will be penalized with a cost of 5000. This way, we can keep the nodes that misbehave outside of the computed tree's core.

Higher peaks represent a larger instability in the network (either of essential core nodes, that have many peers, or multiple leaf nodes). As one can observe, the system quickly restores the tree and brings the topology to a stable status. We note that this system is in production since 2007, being actively used by the community.

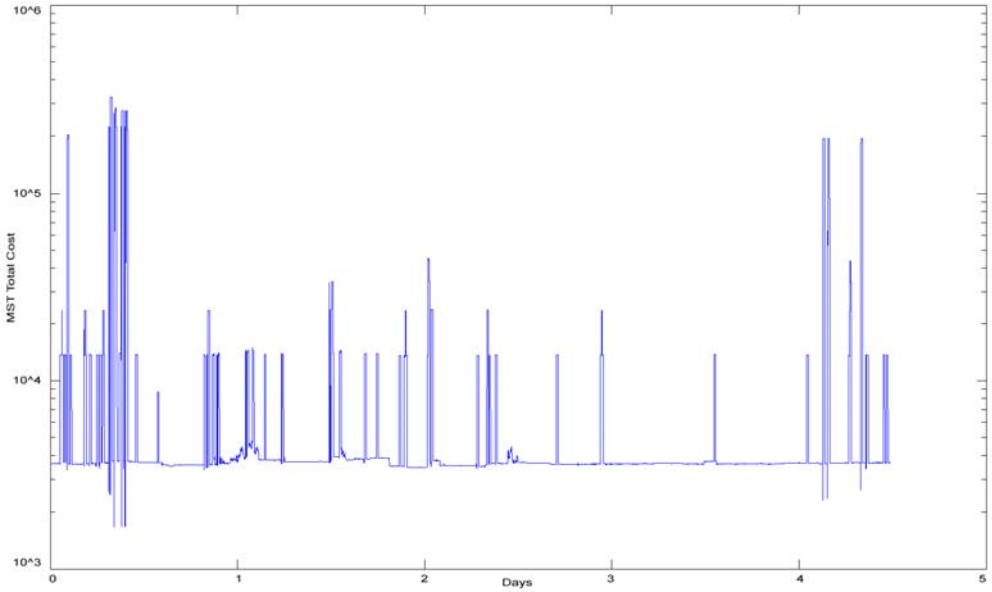

Fig. 11. Evolution of the MST total cost over a period of 5 days

6. Conclusions and future work

In large distributed systems, such as the Data Grids created for the data intensive applications of the HEP experiments, monitoring information is vital for understanding the behavior of the system, the fault conditions and fine-tuning the operation. Developing on the MonALISA framework, we could accommodate advanced features ranging from controlling of remote services to complex high-level clients that can not only monitor, but also drive the distributed system.

Integrating the FDT application brought the flexibility to introduce new data transfer handling policies that help optimizing the data transfers on a different perspective, besides the raw efficiency of the transfer protocol. It also allowed novel developments such as dynamic bandwidth adjustment at the application level, an elegant solution to problems of the classic time-based scheduling approach.

We demonstrated the utility of the *Global Connectivity Optimizer* in the EVO videoconferencing system where it runs in production since 2007. The overlay topology used by the videoconferencing system, which is maintained in a permanent optimum shape by our framework, interconnects hundreds of users, participating in tens of daily conferences. The lack of any major outage stands as proof for its effectiveness.

We have demonstrated the functionality of our framework in a real-life environment, such as the worldwide distributed ALICE Grid, proving that our proposed goals can be achieved.

Since the *Global Connectivity Optimizer* is already in production, our future research interest is focused more towards the *Transfer Scheduler Optimizer*. In this area, we plan several developments, which include: adding support for rescheduling of the requests, implementing the ideas for *Optimizer*'s replication, developing a programmatic interface to the system based on web services, integrating the two policies for handling requests and investigating more complex scheduling algorithms.

REFERENCES

[1] The Large Hadron Collider, http://www.cern.ch/LHC
[2] *Ann Chervenak*, *Ian Foster*, *Carl Kesselman*, *Charles Salisbury*, *Steven Tuecke*, The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets, http://www.globus.org, 1999
[3] Dynamic Resource Allocation Controller (DRAC), http://www.nortel.com/drac
[4] Dynamic Resource Allocation via GMPLS Optical Networks (DRAGON), http://dragon.east.isi.edu
[5] On-demand Secure Circuits and Advance Reservation System (OSCARS) http://www.es.net/oscars/
[6] The File Transfer Service, http://egee-jra1-dm.web.cern.ch/egee-jra1-dm/FTS/default.htm
[7] T. *Kosar, M. Livny*, Stork: Making data placement a first class citizen in the Grid, ICDCS, 2004
[8] Fast Data Transfer: http://monalisa.cern.ch/FDT
[9] *I.C. Legrand, H.B. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, M. Toarta, C. Dobre,* MonALISA: An Agent based, Dynamic Service System to Monitor, Control and Optimize Grid based Applications, CHEP, 2004
[10] *Thomas H. Cormen et. al.*, Introduction to Algorithms, Second Edition, McGraw-Hill, 2001
[11] *C. Cirstoiu, C. Grigoras, L. Betev, A. Costan, I. C. Legrand*, Monitoring, Accounting and Automated Decision Support for the ALICE Experiment Based on the MonALISA Framework, HPDC07, California, USA, 2007
[12] *S. Bagnasco, L. Betev, P. Buncic, F. Carminati, C. Cirstoiu, C. Grigoras, P. Mendez Lorenzo, A. Peters, F. Rademakers, P. Saiz*, AliEn2: The ALICE Grid Environment, CHEP07, Canada, 2007
[13] EVO Videoconferencing system, See http://evo.caltech.edu
[14] *D. Adamczyk, D. Collados, G. Denis, J. Fernandes, P. Galvez, I. C. Legrand, H. Newman, K. Wei*, Global Platform for Rich Media Conferencing and Collaboration, CHEP03, California, 2003
[15] *M. Toarta, C. Cirstoiu*, Monitoring and controlling applications using MonALISA. Case study: VRVS, RoEduNet International Conference, Timişoara, 2004