VU Research Portal

Optimizing the execution of many-task computing applications using in-memory distributed file systems
Uta, A.

2017

document version
Publisher's PDF, also known as Version of record

Link to publication in VU Research Portal

citation for published version (APA)
Uta, A. (2017). *Optimizing the execution of many-task computing applications using in-memory distributed file systems*. [PhD-Thesis - Research and graduation internal, Vrije Universiteit Amsterdam].

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

E-mail address:
vuresearchportal.ub@vu.nl

Download date: 11. Jan. 2025

4

MemEFS - a Network-Aware In-Memory Distributed File System

Abstract

Scientific domains such as astronomy or bioinformatics produce increasingly large amounts of data that need to be analyzed. Such analyses are modeled as scientific workflows - applications composed of many individual tasks that exhibit data dependencies. Typically, these applications suffer from significant variability in the interplay between achieved parallelism and data footprint. To efficiently tackle the data deluge, cost effective solutions need to be deployed by extending private computing infrastructures with public cloud resources. To achieve this, two key features for such systems need to be addressed: *elasticity* and *network adaptability*. The former improves compute resource utilization efficiency, while the latter improves network utilization efficiency, since private clouds suffer from significant bandwidth variability. This work extends MemEFS, an in-memory elastic distributed file system by adding *network adaptability*. Our results show that MemEFS' network adaptation policy achieves up to 50% speedup compared to its network-agnostic counterpart.

The contents of this chapter are under submission in the Future Generation Computer Systems journal, and have been slightly modified to improve readability.

4.1 Introduction

Many-task computing applications are typically deployed on private computing infrastructures. In such setups, due to the inherent application structure, the challenge is to achieve high-performance, and resource-efficient storage solutions. In our previous work [110], we have presented an elastic in-memory distributed file system for running MTC applications on clusters. MemEFS drastically improves resource utilization efficiency for such platforms.

Limiting the elastic approach presented previously to private computing infrastructure capacity, however, is ill-suited for the rapid increase in the data volumes produced by typical scientific applications [42]. Ideally, to achieve cost-efficient in-memory storage solutions, the private computing infrastructure would be augmented by means of on-demand, public cloud resources.

Here, a new type of problem appears: in contrast to private computing infrastructure, many studies [95, 11] point out that in public clouds the network performance is impacted by large degrees of variability - due to virtualization, colocation and congestion overheads [56, 13].

This chapter introduces a data distribution policy proportional to network capabilities that aims to better utilize compute and network resources. The key insight of the proportional policy is that the interplay between compute-to-storage ratio and available bandwidth highly impacts the overall system performance. We implement this policy in **MemEFS**, our locality-agnostic in-memory elastic storage system [110]. In contrast to network-agnostic setups, MemEFS is able to adapt to its current network infrastructure, thus largely improving the bandwidth resource utilization.

The contribution of this chapter is two-fold:

- We introduce a network-aware mechanism that enables MemEFS to seamlessly access resources located across networks without bandwidth guarantees.
- We demonstrate the efficiency of our design using a variety of real-world and synthetic scientific workloads. We show that MemEFS' network adaptability mechanism reduces execution time by up to 50%.

This chapter is organized as follows. Section 4.2 sketches the background and design of our work. Section 4.3 describes the evaluation results, Section 4.4 discusses related work and we draw conclusions in Section 4.5.

4.2 MemEFS Network Adaptation

In this section, we give a short overview of MemEFS, our elastic, in-memory distributed file system for MTC applications. Further, we argue that for migrating MemEFS to public compute clouds one needs a network adaptation mechanism to take advantage of the bandwidth imbalances of such platforms. Then, the MemEFS adaptation mechanism is explained in detail.

4.2.1 MemEFS

Previously, we introduced MemEFS [110], an elastic in-memory runtime distributed file system that highly improves resource utilization efficiency for scientific workflows. MemEFS stores data in main memory and is able to saturate premium network (InfiniBand, 10G Ethernet) bandwidth [112]. Using a two-layer consistent hashing [57] scheme, MemEFS spreads data evenly across cluster reservation nodes. Hence, MemEFS achieves balanced storage and network traffic.

MemEFS implements consistent hashing through a *two-layer hashing scheme* that maps file stripes to *partitions* and then partitions to nodes. We assume it is preferable to organize data in a manner that permits moving small numbers of large objects (partitions) rather than large numbers of small objects (file stripes). Better performance is achieved when transferring larger objects since fewer data transfers are needed, and hence we minimize the latency and maximize the bandwidth utilization. With this argument in mind, each node holds multiple partitions, such that, when reconfiguring the file system, we migrate partitions, thus avoiding rehashing the file stripes.

Throughout the application runtime the number of partitions is constant. The total number of partitions sets the upper bound on the number of nodes to which the elastic distributed file system can scale out to. The size of each partition is limited by the total amount of memory the nodes have. Thus, when running on a small number of nodes with many partitions, the partition size will be small. When scaling out to a larger number of nodes, a subset of the partitions will be migrated to the newly added nodes, allowing all the partitions to grow in size. MemEFS computes the number of partitions each node stores after each reconfiguration by adapting the algorithm proposed in Y0 [37].

4.2.2 Network Awareness

With a highly distributed design, file systems like MemEFS are not robust to network variability. While in a private infrastructure setup this aspect is downplayed by the intrinsic performance isolation of the execution environment, the situation is radically different in public clouds. Figure 4.1 plots the results of an exhaustive study [11] which showed that the network performance is highly variable in public clouds. Such an imbalance in the bandwidth observed by MemEFS nodes would lead to a general slowdown of the system, as faster connected nodes will be reduced to the speed of the slower connected ones. Thus, adding network adaptability to a file system like MemEFS also affects the design of its data distribution and load balancing mechanisms. In this work, we introduce a network-aware data distribution policy in the design of MemEFS.

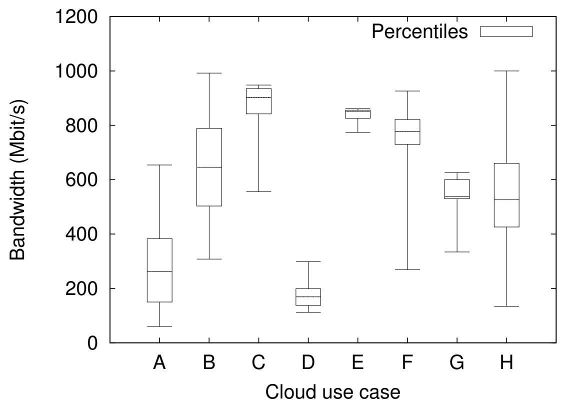


Figure 4.1: Bandwidth distributions for A-H cloud network bandwidth distributions. 1st-25th-50th-75th-99th percentiles. Data points courtesy of authors of [11].

As typical private clusters have homogeneous networks, nodes of the same type receive the same number of MemEFS partitions. In public clouds, when dealing with highly heterogeneous links, we assume that it is beneficial to distribute MemEFS partitions in such a way that nodes with more bandwidth host more partitions.

In a cloud setup, we start by measuring the available bandwidth of the MemEFS virtual machines. Next, we adapt again the Y0 algorithm to compute the numbers of partitions per virtual machine according to the bandwidth capacity; Equation 3.2 becomes:

$$c_v = \frac{Bandwidth(v) \times n}{\sum_{u \in Nodes} Bandwidth(u)} \tag{4.1}$$

The bandwidth distributions shown in Figure 4.1 are the results of benchmarking network traffic between virtual machines in the respective clouds over relatively coarse periods of time (hours-days). These results provide clear upper and lower bounds of achievable performance when two virtual machines communicate in a cloud. However, due to the uncertainty of the underlying conditions (e.g. networking activity bursts of other cloud customers' virtual machines colocated on the same physical infrastructure as MemEFS), the bandwidth variation granularity could be much finer (e.g. seconds, minutes).

To counteract this behaviour, we have implemented a *bandwidth monitoring* process in the MemEFS central manager. The monitoring process collects achieved bandwidth information from the worker nodes, and, based on a configurable time window, takes reconfiguration (i.e. data migration) decisions in case some nodes suffer from network performance degradation. As migrating data to rebalance the system (according to bandwidth capacities) means that the running application is suspended, a small time monitoring window could translate to many reconfigurations of the system, resulting in performance loss.

Therefore, a trade-off between the time monitoring window period and possible loss of performance due to fine granularity bandwidth variations needs to be considered. Moreover, when considering such trade-offs, the runtime of the application needs to be taken into account: for shorter applications (tens of minutes), many reconfigurations could result in unwanted slowdowns; for longer running applications (hundreds of minutes, hours) making MemEFS more reactive, could result in important speedups. As our target applications fall in the latter category, for MemEFS we implemented a bandwidth time monitoring window of 15 minutes. Therefore, when MemEFS' central manager notices that certain nodes' bandwidths have degraded by more than 20% compared to the previously known values, it triggers a reconfiguration. Considering shorter running applications or finer grained network variability intervals is outside the scope of this thesis, and is therefore left for future work.

4.3 Network Adaptability Evaluation

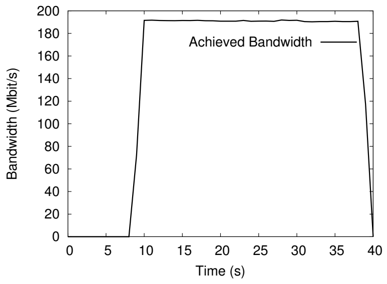
To show the efficiency of MemEFS adapting to cloud-like network conditions, we first assessed the optimization potential of the network-adapted MemEFS and then studied its behavior on real-world workflows. As network variability would affect the overall application runtime, we evaluate in a cloud setup the performance of MemEFS in terms of the application runtime.

4.3.1 Cloud Experimental Setup

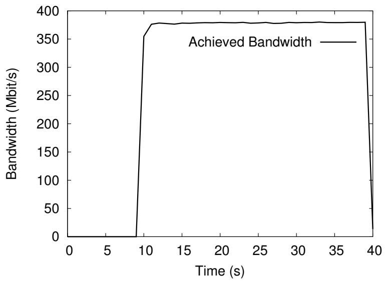
In [11], Ballani et al. present the network variability in eight real-world cloud data centers. We use these eight network bandwidth distributions to emulate the real-world network bandwidth variability in our controlled Open Nebula [77] environment installed on DAS4 [22]. We chose this approach over directly using a public cloud to create a controlled environment.

To emulate these eight datacenter network bandwidth distributions [11], we use *the hose model* [39] to control bandwidth in our Open Nebula deployment. This model is a simple virtual network overlay able to limit VM-to-VM traffic according to the user specification. To implement this bandwidth limitation model we use the Linux *tc* tool [47]. For our experiments, we use 32 VMs that each have 8 cores and 20GB of memory. Their allotted bandwidth capabilities follow the distributions reported in [11], shown in Figure 4.1.

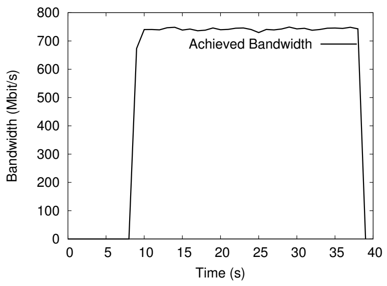
Figure 4.2 shows the bandwidths achieved by our hose-model limitation mechanism. Figures 4.2a - 4.2d show the achieved bandwidth of a virtual machine when its network traffic is limited to $bw \in \{200, 400, 800, 1000\}$ Mb/s. We notice that the achieved values are, in practice, slightly lower than the theoretical limitation. However, this is an expected behaviour and it is important to notice that the difference is always below 10%.



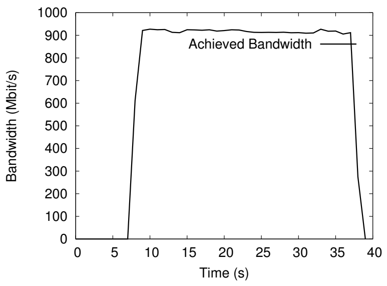
(a): Bandwidth limit = 200 Mbit/s



(b): Bandwidth limit = 400 Mbit/s



(c): Bandwidth limit = 800 Mbit/s



(d): Bandwidth limit = 1000 Mbit/s

Figure 4.2: VM achieved bandwidth when network link is limited to $bw \in \{200, 400, 800, 1000\}$.

4.3.2 Network Adaptability Experiments

We first study the optimization potential of the network-adapted MemEFS by running an I/O-intensive microbenchmark on each setup. The workload is composed of 1000 tasks, each writing 100MB of data to MemEFS. We chose this microbenchmark configuration due to its behaviour of fully saturating the network bandwidth. We noticed in Section 2.4, when evaluating the MTC Envelope, that writing larger files in our file system achieves higher bandwidths than smaller files. Therefore, we use this benchmark as a *stress test* to verify our network-adaptation mechanism for MemEFS.

We compared the performance achieved by the network-adapted MemEFS to the network-agnostic version, that distributes partitions evenly across nodes. Figure 4.3 shows the evaluation results.

The results are consistent with the bandwidth distributions depicted in Figure 4.1. The distributions *A, D, F, G, H* exhibit more network bandwidth variability. The network-adapted MemEFS is therefore able to leverage this variability and improve performance. The *B, C, E* distributions indicate much more stable network conditions, thus adapting MemEFS to their bandwidth characteristics can intrinsically deliver a much smaller performance gain.

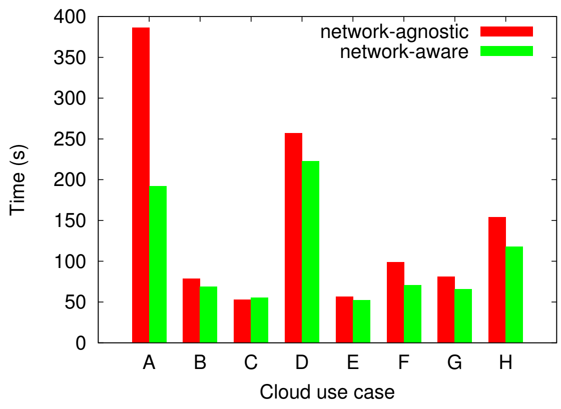


Figure 4.3: Network-Adapted MemEFS vs. Network-Agnostic MemEFS on 32 VMs running an I/O intensive benchmark (lower is better).

Table 4.1: Workflow Characteristics

Application	# Tasks	Input Size	Peak Storage
Montage	39472	13GB	320GB
BLAST	3072	57GB	192GB

To further analyze our network adaptation policies, we selected two real-world applications, *Montage* and *BLAST*. Their parallel stages are representative for a wide range of the typical workload characteristics spectrum. First, *mProjectPP, mBackground* stages of *Montage* are CPU-bound, while *mDiffFit* is I/O bound. Next, although *BLAST* tasks are CPU-bound, they also show moderate memory and I/O utilization. Considering runtime, the *Montage* tasks are short (order of seconds), while *BLAST* tasks are longer (tens of seconds to minutes). Considering intermediary data size, while *Montage* generates small files (1-4MB), *BLAST* deals with much larger files (hundreds of MB). Therefore, we consider running these two applications sufficient to validate our network-adapted MemEFS. The workloads' characteristics are shown in Table 4.1. When reporting the runtime of each experiment, we compute the full makespan of the applications (i.e. since the data stage-in starts until the last task of the workflows has finished).

Given the limited size of our cloud setup, we have scaled down the *Montage* and *BLAST* applications such that their generated data could fit in 32 VMs, each equipped with 20 GB memory. Furthermore, when the bandwidth distribution is highly heterogeneous (e.g. cloud setups A, B, F, H), some VMs may store much more data than others in the network-adapted setup. As a consequence, we had to choose application sizes that do not generate more than 20 GB of data per VM in the most extreme bandwidth distribution setups.

Figure 4.4a shows the *BLAST* runtimes when using the network-aware and the network-agnostic MemEFS systems in each emulated cloud network bandwidth distribution. The results clearly show that for the more bandwidth-imbalanced setups (A, B, D, F, G, H), the network-aware MemEFS outperforms the network-agnostic MemEFS.

Figures 4.4b and 4.4c show the bandwidth in and bandwidth out utilization: the network-aware setup consistently achieves better bandwidth utilization than the network-agnostic setup. This is because in the network-agnostic setup, fast nodes are slowed down by the slow node and cannot fully utilize their bandwidth capacity. In the network-aware setup, the fast nodes hold more MemEFS partitions and thus more I/O operations are directed to them.

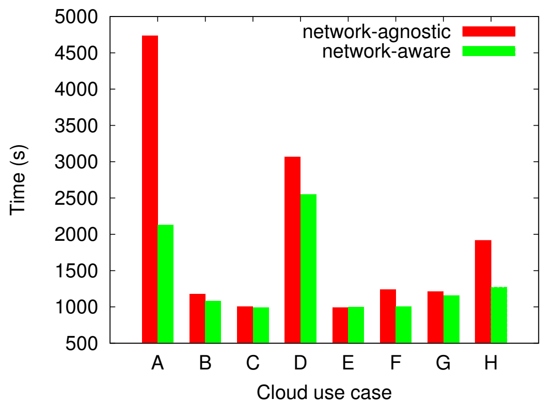
Figure 4.5a shows the runtimes obtained for the *Montage* workload. Figures 4.5b and 4.5c show the bandwidth utilization when running the network-aware and network-agnostic versions of MemEFS on the emulated cloud setups. The more bandwidth-imbalanced setups lead to much better performance when using the network-aware MemEFS. This is a direct consequence of the network-adapted MemEFS making better use of the available bandwidth by distributing more data to the faster nodes.

For both workflows, we notice that although the bandwidth utilization is improved in the network-aware setup, we do not reach full utilization. The explanation for this behavior is twofold. First, *BLAST* tasks are not I/O-bound. Since MemEFS I/O is done through the network, *BLAST* tasks are unable to saturate the bandwidth. Second, in the *Montage* case, the parallel stages (*mProjectPP, mDiffFit, mBackground*) are intertwined with (long-running) sequential stages (*mImgTbl, mConcatFit, mBgModel*). These synchronization points decrease the overall network utilization.

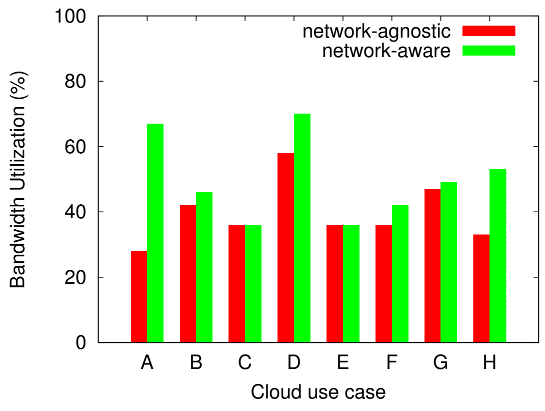
For the clouds that exhibit less network variability (B, C, E), the network adaptation mechanism of MemEFS cannot achieve better bandwidth utilization since the partition-to-node mapping is similar to the network-agnostic setup. Furthermore, the network utilization achieved in this setup is close to 40% due to application characteristics: *BLAST* tasks are not I/O bound, while *Montage* has synchronization points that decrease the overall network utilization.

4.3.3 Discussion

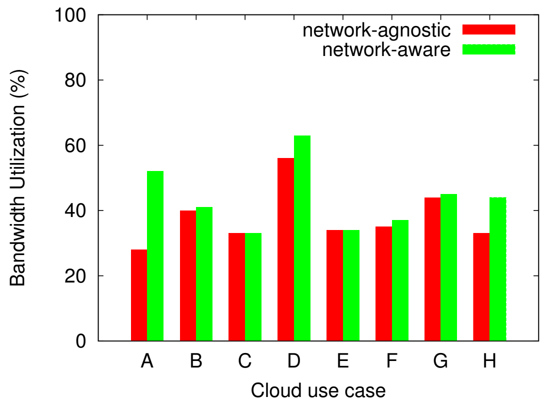
When migrating scientific applications to public clouds, our experiments show that it is imperative to adapt MemEFS to the underlying network characteristics. Using real-world cloud data center bandwidth distributions [11], we evaluated MemEFS' network adaptation mechanism on two real-world applications that exhibit different structural characteristics and runtime behaviors. Our results show that the MemEFS' network adaptation mechanism greatly improves performance in cloud setups suffering from high bandwidth variability. In such scenarios, MemEFS' adaptation policy improves the application performance by up to 50% through judicious use of the available network bandwidth.



(a): Total execution time (lower is better).

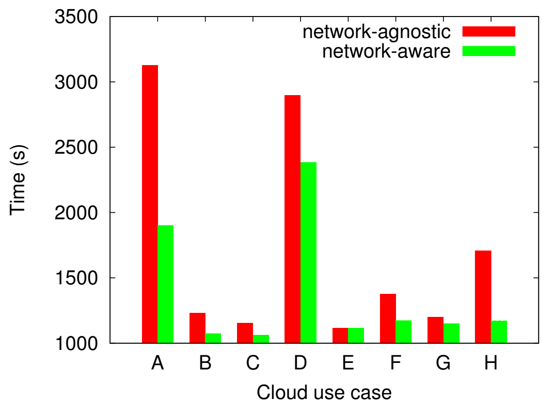


(b): Inbound Bandwidth (higher is better).

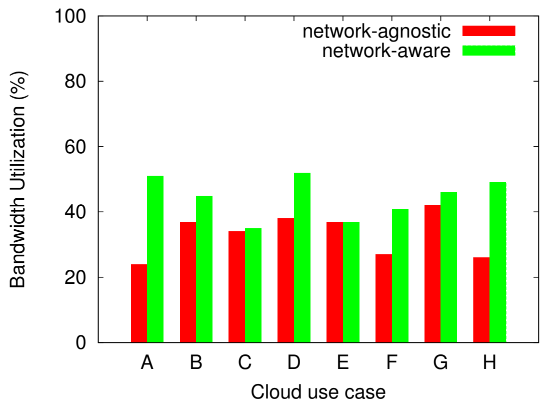


(c): Outbound Bandwidth (higher is better).

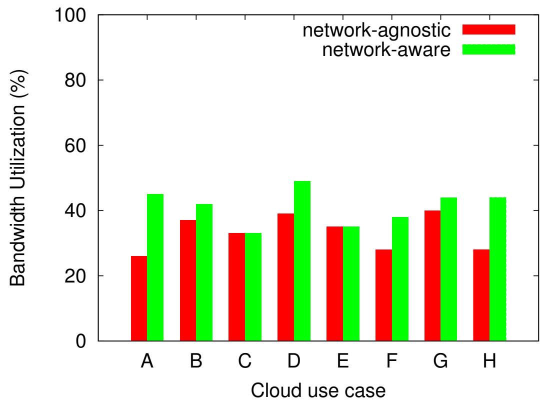
Figure 4.4: BLAST on 32 VMs executed using network-agnostic MemEFS and, respectively, network-adapted MemEFS.



(a): Total execution time (lower is better).



(b): Inbound Bandwidth (higher is better).



(c): Outbound Bandwidth (higher is better).

Figure 4.5: Montage on 32 VMs executed using network-agnostic MemEFS and, respectively, network-adapted MemEFS.

4.4 Related Work

We identified several studies that improve application performance when the underlying compute resources suffer from bandwidth variability. The main difference between our approach and these studies is that we target the distributed file system level. Therefore, our approach is more generic and transparent to the application and scheduler.

In [84], the authors propose the use of a *Software Defined Network* (SDN) to achieve a bandwidth-aware scheduler for Hadoop. They utilize the link measuring and bandwidth setting capabilities of an SDN to distribute data and tasks in such a way that the makespan of a MapReduce job is minimized. Another framework [58] that adds network awareness to Hadoop considers multi-cluster Hadoop setups. Because inter-cluster bandwidth is often lower than intra-cluster bandwidth, they account for this in their scheduler and achieve good performance in terms of makespan. *EHadoop* is another framework that also takes into account the network usage of MapReduce jobs when scheduling [119]. It decouples data from computation by having two separate clusters. The bandwidth between the two clusters is variable. By performing online profiling of task network usage and completion time, EHadoop keeps job completion time stable when faced with different network topologies.

In [64], the authors describe a task scheduler for independent tasks that incorporates bandwidth knowledge to schedule tasks on resources. Tasks are scheduled on VMs with different bandwidth capabilities. It is not immediately clear, however, if the bandwidth requirements of the individual tasks are known beforehand. Assuming available bandwidth is known, but variable, the authors from [18] propose a DAG workflow scheduler that minimizes makespan using fuzzy optimization techniques. It can handle workflows that have inter-task data dependencies, such as Montage. It assumes data transfer information between tasks is known beforehand.

4.5 Conclusions

With the rapid increase of data volumes generated by scientific domains such as bioinformatics or astronomy ("data deluge"), we expect scientific workflows to outgrow the (memory) storage capacities of private clusters. As a consequence, clusters will need to be augmented by means of public cloud computing infrastructure. However, as many studies point out, such platforms are plagued by large bandwidth variability due to colocation and virtualization overheads.

In this chapter, we have introduced a network bandwidth adaptation mechanism for MemEFS, our elastic in-memory distributed file system for many-task computing applications. Our evaluation show that it is imperative to adapt the storage layer to the underlying network, as otherwise the application would observe a severe performance penalty. To overcome this performance penalty, MemEFS' network adaptation mechanism takes advantage of network variability and increases performance by up to 50%. Our experiments show that the larger the network variability is, the more MemEFS outperforms the network-agnostic approach.