6

General Conclusions

Initially, scientists used large-scale computing infrastructures to run simulation-based compute-intensive applications. Following Moore's law, the increase in compute power has greatly benefited such applications, as bigger problems could be tackled more efficiently and solved faster. Then, as scientific domains evolved and researchers collected large amounts of experimental data, a new data-driven (data-intensive) research paradigm has emerged. Nowadays, the time to access the large volume of data to be analyzed can be orders of magnitude higher than the computation time itself. This is because the rate at which storage speed increases is greatly lagging behind the rate at which computational power increases. This imbalance between compute and storage speed greatly limits application performance and scalability.

To make matters even more difficult, various scientific analyses are expressed as a special class of applications - *many-task computing*. This class comprises computations that are composed of loosely-coupled, short-lived tasks that communicate through file system operations (i.e. by means of files). Therefore, when running such applications, the storage system has a dual role: not only does it store application input and output, but also relays communication data. This additional role adds more pressure to the storage layer, limiting application performance and scalability even further.

To improve performance and scalability, in this thesis we have investigated how a storage system could be tailor-made for many-task computing applications. To achieve this, we have proposed storing communication data in main-memory rather than traditional, disk-based storage. In our approach, the compute nodes running application tasks share their memory in a common pool of fast storage exposed as a distributed file system.

Even though, traditionally, network was slower than disk, due to recent advances in network technology we consider premium networks (e.g. InfiniBand, 10G Ethernet) the main catalyst for achieving high performance in in-memory distributed file

systems. Current network bandwidth has dramatically outgrown disk bandwidth and is approaching the performance offered by main memory.

6.1 Thesis Contributions

The contributions presented in this dissertation are four-fold and concern both application- and platform-specific design decisions. The former focus on improving many-task computing applications' performance by storing their runtime generated data on in-memory distributed file systems. The latter focus on improving resource utilization of such systems when deployed on private clusters and public clouds.

In Chapter 2 we presented the design of our in-memory distributed file system, MemFS. As opposed to locality-based state-of-the-art in-memory storage systems, MemFS is locality-agnostic. By spreading data evenly across compute nodes' memory, MemFS is able to overcome locality-induced bottlenecks: imbalanced storage and network traffic. Due to its locality-agnostic design, MemFS achieves good vertical and horizontal scalability when deployed on both private clusters and public clouds with fast premium networks.

In Chapter 3 we presented MemEFS, an elastic extension of MemFS. We noticed that many-task computing applications exhibit a large variability of both the achieved parallelism and data footprint. The former is given by the inherent application structure: large parallel stages are followed by single-task data partitioning and aggregation stages. The latter is induced by the lifespan of the files that are produced during runtime: a file is only stored as long as there are unexecuted tasks that require it as input. This behaviour puts additional pressure on the developer or scientist that runs such applications, as the compute nodes need to be overprovisioned such that the system can accommodate the maximum amount of data needed during runtime. Moreover, due to the variability of the achieved parallelism, overprovisioning results in severe resource under-utilization. To overcome this behaviour and improve resource utilization, MemEFS' elasticity ensures that the application uses only as many resources as needed during runtime. This shrink-grow mechanism largely improves resource utilization and eliminates the need for overprovisioning.

In Chapter 4 we prepared MemEFS for deployment of public clouds. With the ever-growing volumes of data needed to be analyzed by scientists, we expect even larger amounts of data to be produced during the runtime of many-task computing applications. Therefore, we expect systems like MemEFS to soon outgrow the (memory) capacities of private clusters. To reduce operational costs, we expect private infrastructures to be augmented by means of public cloud resources. However, such infrastructures do not offer performance isolation and are known to suffer from significant network bandwidth variability. As private clusters achieve stable network performance, MemEFS distributed data equally among compute nodes. In a cloud setup this would result in performance degradation as fast nodes would be slowed down by the slow ones. As a consequence, to prepare MemEFS to be deployed

on public clouds, we have designed a network-aware data placement scheme. This scheme significantly improves MemEFS performance on clouds.

In Chapter 5 we introduced MemFSS, an extension of MemFS, that performs memory scavenging through software resource disaggregation. The de-facto platform for running large scale applications, clusters, exhibit large amounts of underutilization for two resources: memory (up to 50%) and network. This is because typical cluster applications, such as Hadoop MapReduce or MPI computations are highly optimized for CPU utilization. To achieve high performance, the two resources that MemFSS relies on are memory and network. To improve cluster resource utilization, we restrict many-task computing applications to run on a smaller set of MemFSS *own nodes*. The storage space is therefore extended by scavenging for available memory in other tenants' reservations (*victim nodes*). In this way we reduce the resource requirements of our deployment, and make use of the underutilized resources in the cluster. Our technique induces negligible overhead in typical *victim* applications (e.g. Hadoop, Spark, MPI), while the resources needed to run many-task computing applications are dramatically decreased. As a consequence, total cluster resource utilization is drastically improved as multiple types of applications, with different resource utilization patterns, are able to run in a symbiotic fashion.

6.2 Future Directions

In this thesis we have showed how our in-memory distributed file system is able to improve the execution of many-task computing applications when running on either clusters or clouds. We believe that our comprehensive study of the performability aspects of such systems also opens promising future research directions. We envision two different categories of future work: (i) further (performance) optimizations and fault-tolerance policies; and (ii) the implications and applicability of our work in other fields.

We believe that the performance of our in-memory distributed file system could be further improved by using a specialized data store instead of memcached or redis. A tailor-made data store could make use of more lightweight network protocols, such as RDMA. This could further improve bandwidth and reduce latency. Moreover, the FUSE file system interface to applications could be replaced with a platform that exhibits less latency, such as a userspace library file system or a kernel module.

Further efforts could also be invested into improving reliability and fault-tolerance. Since main memory is volatile, a crash of a compute node could result in loss of important data. Also, as replication is a policy that would severely reduce storage space, as memory is still a scarce resource compared to disk sizes, other techniques need to be considered. Even though erasure codes are a promising approach, encoding/decoding would consume extra CPU cycles and network bandwidth, which could degrade application performance. An alternative could be a checkpointing-

recomputation approach. Based on the application DAG, the scheduler could choose to checkpoint certain tasks and their outputs, such that when a crash occurs the number of recomputed tasks would be minimized. Considering recent advances in NVRAM technology, another interesting approach would be to map our in-memory distributed file system to non-volatile memory rather than main-memory.

Our memory scavenging approach could have important implications when applied to other data processing systems or platforms. Since clusters and data centers are severely under-utilized in terms of memory and network, we believe that systems such as Spark, stream processing platforms, or graph processing platforms could improve performance or increase in-memory storage space by scavenging for available memory. Furthermore, in cloud data centers, (unutilized) disaggregated memory could be used to cache hot virtual machine images for speeding up the booting process.