Developing a Cloud-based Virtual Machine

Alexander Schmidt[1]
InRule Technology, Inc., Chicago, IL

Dmitry Tkach
Schneider Electric, Fresno, CA

Abstract

This paper discusses the approaches for a Virtual Machine optimized for executing in Cloud environment. The optimization concerns with I/O operations, memory management, and job scheduling. We address the problem of development of software using the provided Cloud environment. Our first contribution is re-modeling of existing frameworks for traditional distributed storage systems. The second contribution is introduction of the job scheduling framework as a standard component in the system for developing event-based solutions; since the resulting framework is generic, we show how to use it in managing the data through sample problems. Consequently, we contribute to the design of a library of proof-based developed algorithms.

Keywords: Cloud computing, internal cloud, virtual machine, cloud framework

1 Introduction

The paradigm of the virtual machine has changed significantly recently with the introduction of the cloud computing and the availability of various services to the developer via public interfaces. [1]. Cloud providers offer their APIs which can be installed to the framework of the developer's choice and make it available for other components of the application through a number of protocols such as REST, SOAP, and others. The plethora of this is to bring the processing into a unified whole such as unstructured binary storage, relative database storage, background workers, and platform web services. The unstructured binary storage adds a capability to store the data in virtually unlimited cloud space over the Internet and compliments the legacy file system and network APIs such as NFS and DFS. This approach brushes aside a possibility to organize the data in the internal clouds which are running within the network boundaries.

The existing distributed memory systems are known for a long time as citizens of the high-performance computing (HPC) in which communication between each node of cluster is established with point to point links or other hardware. A key factor of scalability of such systems is determined by the network topology. The next most crucial problem is how distribution of the shared memory (DSM) blocks is organized. Obviously, fast networks and efficient algorithms are vital in the design of such systems. Examples of this class of systems include Kerrighed, OpenSSI, and Terracota.

[1] Corresponding author: A. Schmidt (me@alexschmidt.net)

The job scheduling system provides infrastructure to define workflows, manage chunks of work, and retrieve the results. These systems usually work in an unattended mode and have management and monitoring tools for tracking the health of the system and intervene in the process, if needed. The job scheduling system is a rebranded name known since Mainframe times as a batch processing which started with the popular JCL on IBM machines. Modern implementations are based on the open source and proprietary specialized frameworks and they are the fundamental blocks for the Service oriented architectures (SOA). Examples of such systems include IBM Tivoli, Sun Grid Engine, and Apple Xgrid. Most of these systems are capable of providing the job scheduling services but are difficult to integrate with JRE or .Net virtual machines. The original intent of the job scheduling systems did not incorporate the possibility for integration with clouds and executing jobs in cloud-aware environments.

To facilitate the idea of bringing the different pieces into a common framework, in the Section 2 of this paper the core components are described in detail and compared further with state-of-the-art solutions through samples in Section 3.

2 Virtual machine components

Cloud computing environment provides scalability, performance optimization and accommodates emerging technologies as you go. Being able to running the applications in the cloud is an important aspect of cloud computing. It's possible to provide cloud-based services that they can be used by managed applications developed for traditional environments. In the next section we would like to introduce a concept of the cloud-based distributed file system. The concept will provide an overview of various aspects of traditional file systems in a view of cloud awareness.

2.1 A cloud-based distributed file system concepts

One advantageous application of the cloud computing technologies is a support of the distributed file system concept. The common definition of the distributed file system is that it allows accessing files on the remote cluster nodes in a way that is similar to working on the host computer. It makes easier development, deployment and configuration of applications in the heterogeneous environment.

It appears that there is high demand to have a file system implemented in the cloud in a layer above the system layers providing common standard interface for cloud computing components. It will be beneficial to provide such robust interface to simplify the design of software targeting cloud systems. The design of the cloud file system is based on the data partitioning allowing distributing portions of data blocks across the file system. In order to implement the file system in platform-agnostic way it is necessary to follow elaborative methodologies [4] which are extremely complex. Therefore, a solution is to move parts of the file system to user mode and to use a runtime environment for their logic such as Microsoft .Net, or Sun Java.

There exists an extensive information about the design file system layers [6]. Here we would like to concentrate on the layers which were adapted to accommodate the needs of the cloud storage. They provide a model of the actual environments where computing units and corresponding file system objects being a part of the cloud are representing the constituting elements of the sys-

tem and developed using Cloud Computing Lifecycle Management (CCLM). We would like to introduce a term Dust through Dust Computing (DCe) and Dust Programming Abstractions (DPa). The Dust Transformation Layer tights together several clouds with different Programming Abstractions. Each Cloud Programming Abstraction is connected by Common Dust Programming Abstraction (CPA). So CCLM has two parts, CCE and CPA, which are optimized for maximizing the amount of data stored on a single partition and for providing high fault tolerance. Other core concepts of the file system interface include directory CRUD functions and functions for working with hard and soft links.

The distributed cloud file system is represented as a balanced tree which is composed of internal nodes and leaf nodes. Each node can be a data block or single computing unit. Each object is assigned a unique key – UUID. By default, UUIDs will be stored as part of the node metadata. For instance, file data blocks which belong to the same directory are grouped together and have the same status under the file system sub-tree. Data block node consists of a list of attributes such as UUID, info-table, and node pointers. In some cases, several types of nodes can be used during the file system mutation such as root nodes, index nodes, and leaf nodes. In order to create a new node, a root node is allocated and inserted to the global file system tree. Same principle can be used for deleting any data node when only a traversing operation is necessary to deallocate the node. Standard journaling file system mechanism should be implemented in order to handle complexity of transactional support for inserting, cloning, deleting, and creating file operations. Basically, each time when an operation on the cloud file system is performed, instead of executing operations directly in file system layer, the transactions are committed into the file system journal.

One of the most complicated tasks from the architectural perspective is data partitioning in the distributed system. The proposed implementation of the distributed cloud file system relies on incorporating generic object-oriented concepts using B-tree search as a core structure and on using commonly known algorithms such as linear hashing, value lists, and inverted indexes hashing. Namely, every data file or directory is represented via file system node object with a list of attached attributes.

Constant demands in maintaining the consistency of the file system is in conjunction with slow and fast networks leads to problems of sharing available resources and shadowing data nodes. The typical client-server architecture does not work well in such heterogeneous conditions. The proposed implementation of the file system does not have strictly defined data block sizes such that the data files can have flexible sized data blocks. From the implementation standpoint it does not matter if the data will be manipulated as streams. Each operation is executed in a transaction context. Normally, all operations including updating data will be kept in memory. Like data, we keep cloud environment metadata in memory too. We must be sure that information is distributed between heap memory and file system shared memory, respectively. Each file system referencing to different nodes in a cluster, either Windows or UNIX based, has its own data type which determined by several abstraction layers including network secure interfaces. There are several attributes responsible for maintaining the node structure in the node translation layer.

In order to provide data integrity of data chunks, a checksum mechanism is required. When a client creates a file in the file system, the system computes a checksum of each chunk of the data

file and injects this checksum in a special tree node attribute. Every time when a single node retrieved, the data file is verified. In addition, to meet the goals of a full scale implementation of the file system, it should support asynchronous and synchronous I/O, direct and buffered I/O, which can be provided using a combination of underlying system primitives as well as runtime libraries such as Java NIO. Regardless of the approach, there should be no performance penalty to access blocks on different hosts; all operations should be efficient in terms of disk utilization and storing and retrieval time of the attribute data injection to the file system tree structure. One of the major characteristics of the cloud file system is the ability to pass open file descriptor from one process to another which leads to extendable concurrent processing capabilities. The design should guarantee that after passing a file descriptor file it will not be closed and all data will be readable not only by one process but by other processes as well, which can be provided by stream-based pipelines. To avoid logical redundancies or having the same data stored in different places process information should have UUID implemented using process tracing facilities.

In order to guarantee the concurrent processing in the file system, it should support transactional operations in clean, efficient manner. Using the file system transactional mechanisms prevents data inconsistency and allows easy recovering procedures via routings, scheduling, logging and caching frameworks. Special structures will be used to allocate transactional events and sequences such as open, lock, unlock, commit, rollback transactions provided in the transaction management framework. In general, to simplify implementation, multiple component frameworks can be introduced such as transaction management framework, caching framework, configuration framework, and database framework as a non-sql but rather hash table storage, providing services for data block allocation and mapping, file management engine, and, finally, scheduling engine. All these components support high performance-data sharing among nodes. The four key requirements define the integrity of distributed file system and making transactional implementation safe, such as consistency, isolation, durability, and atomicity.

It is noted that to be able to manipulate data stored in the file system, it should be loaded to temporary memory. While it is usually sufficient to utilize local memory for this purpose, it is feasible to implement a cloud-based memory system as a part of the common framework. Let us dive into the mechanics of this concept in the next section.

2.2 A cloud-based memory system concepts

It is vital for a cloud to be able to access the shared memory arrays. The technique called distributed shared memory [3, 12] nicely fits into this paradigm and provides the illusion of local shared memory. Although other models of shared memory were developed, such as hardware assisted physical shared memory [7, 8, 9], its infeasible due to the complexities of internal hardware design and increased amount of additional transistors on die or auxiliary service peripheral circuits needed; the operating systems shared memory is difficult to implement in generic operating systems since it requires a significant changes applied to the internal design of core components. These conclusions lead us to the fecund profundity in the design where shared memory is implemented in yet higher level of software, namely, the virtual machine we are discussing. Figure 1 shows the form of shared memory, implemented by user-mode software.

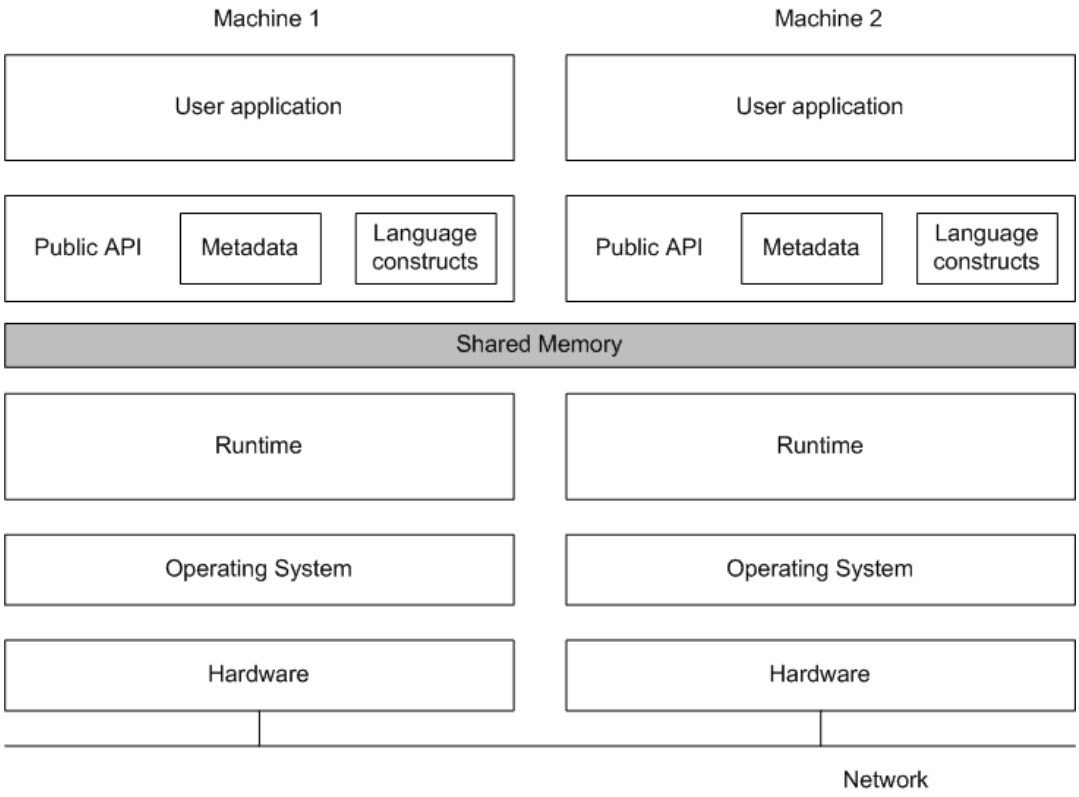

Figure 1 Executing layers for the shared memory implementation

Let us take a look in the details of the software assisted distributed shared memory and how it works.

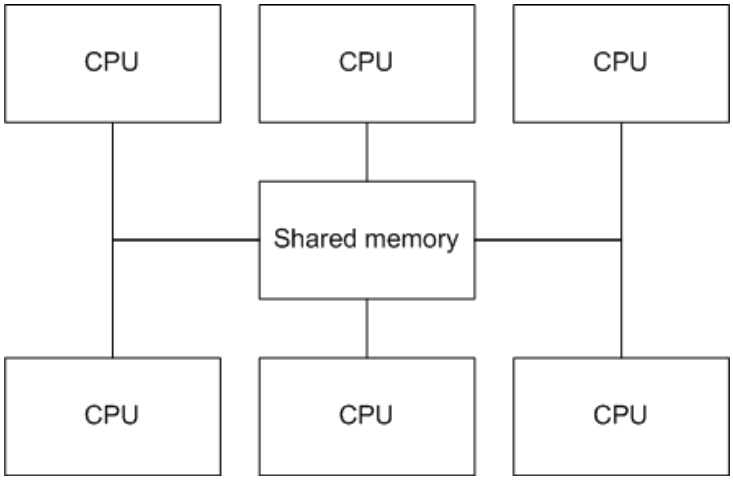

Figure 2 Distributed shared memory virtual memory organization

The distributed shared memory organization is similar to the one shown in Figure 2. A virtual machine has its own virtual memory which is available for CPU operations LOAD or STORE. On system trap, the virtual machine asks CPU that holds the page to unmap the page and sends a re-

quest via network to the owner. When the page arrives, it is mapped to the shared memory and faulting virtual instruction is restarted thus making it appear as it is located in the shared memory and not remote RAM.

The important feature of the distributed shared memory is replication of pages. The read-only page replication is a no-brainer tool that can improve performance. Another possibility is to implement the replication of not only the read-only pages but the non read-only pages with careful consideration on how the just modified pages processing should be done to insure the consistent state of the processes. If the size of the modified pages to be replicated is large, the precautions should be taken for not consuming the entire time in replicating process rather performing useful work. The theory behind optimality and appropriate algorithms are outside of scope of this paper, but can be found elsewhere.

Move over, a difficult problem exists in the distributed shared memory known as false sharing, which occurs when network tied up if one processor has changed a shared variable A, another processor has changed a shared variable B and variables are located on the same page. The replication process is initiated by both processors will occur and page will be travelling back and forth between two machines. This is a difficult problem with no good solution known and usually alleviated by special intrinsic algorithms built into compiler which emits the code.

If we were to implement write page, yet another issue comes into the scene which is archiving sequential consistency. The events in the system occur in certain order which is to be preserved when the replication occurs across machine boundaries. This situation is analogous to the one famous multiprocessor problem when one processing unit attempts to modify a word that is loaded in to multiple local caches. Just as a note that the typical solution there is the CPU about to perform the write puts a signal on the inter-processor bus to discard other CPU's cached copy of the to be modified block. Our solution is to use proved locking mechanism that acquires a lock on a portion of the distributed address space and then perform the read and write operations.

In the next section we provide an overview of the job scheduling system which contributes to the framework along with the concepts presented here.

2.3 A cloud-optimized job scheduling system concepts

Multicomputer scheduling has an analogy to multiprocessor scheduling, when each computation unit waits for a process, loads it in the memory and starts running. To that extend scheduling and related algorithms are similar. The algorithms include various aspects of process scheduling which includes load balancing, a list of ready processes, processor allocation, memory allocation, inter-unit communication, crash recovery, and monitoring. A number of algorithms [12] were introduced for these aspects and their implementation provided in several frameworks, most of which are commercial or specialized for particular applications. The processor and memory allocation algorithms include a graph deterministic algorithm which schedules work based on the matrix with the communication costs between units. A sender initiated distributed heuristic algorithm based on the metric of unit load where an originating unit probes other nodes for their metric and assigns work to the one which reported the load value below certain threshold value; a receiver

initiated distributed heuristic algorithm [10] based on the load of remote units and if the local process has finished, it asks overloaded remote units for work.

The job scheduling system is a special framework that provides a mechanism to sequence the work that the client wants to perform. The system keeps track of input and output and what work will have to be done next. The persistence stores the work queue for the system in case of failure. The scheduler provides a mechanism of work prioritization. The managing unit interacts with different flavors of processors which implement the tasks that job scheduling framework will do.

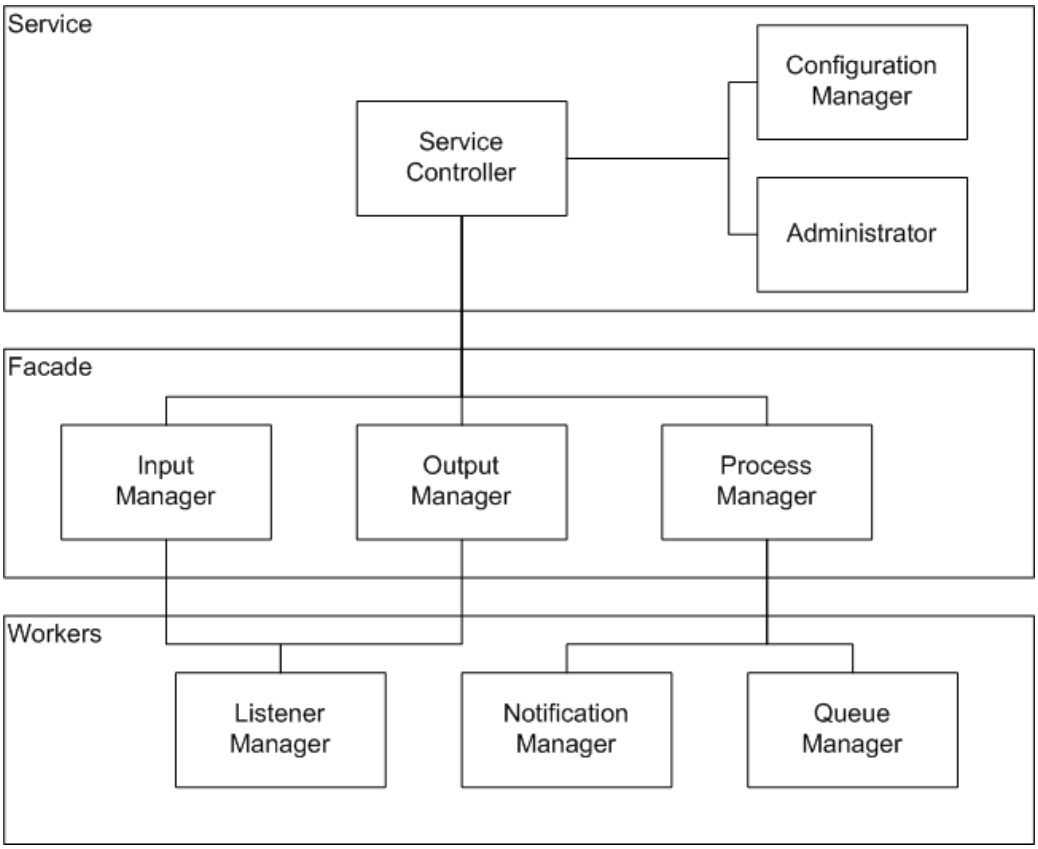

Figure 3 Job scheduling system

A job scheduling system contains several layers as illustrated in Figure 3. It is implemented as a stand-along service running as a background process on the host operating system. The Service Controller class is in charge of starting the major portions of the service, and then monitors health and restarts, if needed, the underlying components. Once started, most of the work is performed by instances of the Input Manager, Output Manager, and Process Manager which are monitored and controlled by the Service Controller class. After the instances of the managers started, the Service Controller continues to monitor changes in the configuration and if a change is noticed, the controller communicates with updated manager and presents new configuration to the manager. The manager then makes adjustments according to the new configuration. This may include changes in network, number of nodes, and hardware changes. Move over, the Service Controller

will allow the Administrator to manage the service which allows starting, stopping, pausing, and resuming the process.

Orchestration of the major portions of system is performed by Process Manager. The manager provides an interface similar to OS service through methods like Start, Stop, Pause, and Resume with additional service methods. After the command in the concrete instance of manager has been issued, one of two situations applies. In many cases the instance must wait until the there is work for it. Work for the manager can be arranged in several ways. The family of classes those are based on the worker, provide a common interface for getting messages arriving through different channels. Listener classes are instantiated and used by the Input Manager. The family of Notification Manager Class provides the envelope to hold the details of an incoming message from different sources. And last major family is the Queue Manager which provides a mechanism to sequence the work that the service will perform. Between the different managers, the work item keeps track of what work will have to be done next.

Following concludes the overview of the job scheduling system designed to be aware of cloud environment and to provide high performance facility for parallel computations. In the previous sections we discussed the components in the implementation of the cloud framework and virtual machine. The first section (2.1) explores implementation of the distributed file system, following by the second design in the section (2.2), which shows how to use the memory system in the distributed environment. The third design, in the section (2.3), shows how we can use the job scheduling system to perform computations.

3 Implementation

In this section, the implementation of the cloud virtual machine API is evaluated and its performance compared with legacy virtual machines performance. The execution engine in the heart of the VM is the most interesting part of the whole system. It contains the runtime system layer which lies upon a host operating system such as Windows, Linux or other and provides platform abstraction services. Next, the services layer contains the component model as well as runtime services, such as job scheduler, state persistence, replicator, compiler performance management, and health monitoring. In many respects, it is a functionality which we refer as a runtime or the virtual execution environment. Besides that, many fundamental mechanisms such as type resolution, module and class loading, stack management and other are implemented here. As a façade, a number of public APIs were implemented to provide interface to the facilities of the underlining functionality of cloud and system packages in a way that has been tailored to provide the services in most effective way and to increase programmer productivity through consistency and quality.

3.1 Cloud virtual machine API

Facilities of various packages in the system are provided to the user applications which will be loaded, compiled and executed in the virtual machine environment. Once compiled, a user application has access to cloud storage which is local, internal or global, facilities of the distributed shared memory, and job scheduling system. Observant reader may have noticed that we didn't really expound how the system packages were implemented as we took an agnostic approach due

to the magnitude of the problem and provided functionality to the user code by exposing the underlining system services.

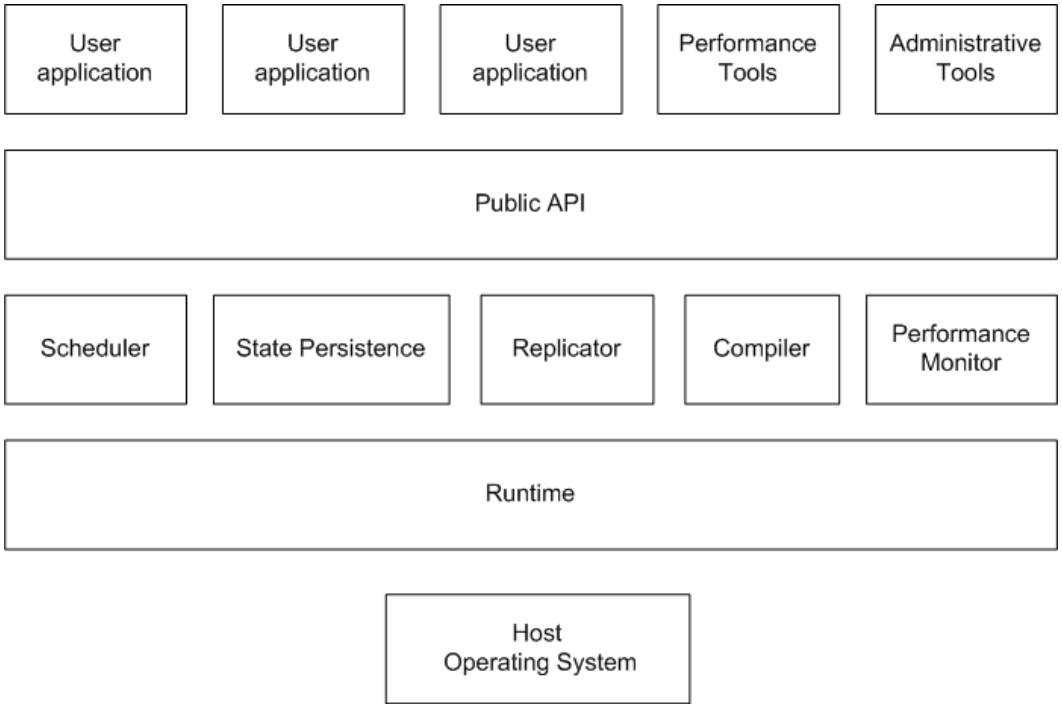

Figure 4 Sample implementation of a Cloud Virtual Machine

In Figure 4 shown a high level diagram of the functional blocks in the sample implementation of the cloud virtual machine which we will review further in this section. Although the detailed discussion of the implementation is outside of scope of this paper we would like to limit our expediency and give a comparative evaluation of the timings obtained for the sample problem using new cloud services provided by Microsoft as Azure platform, Amazon EC and Sun Lustre. To understand how the frameworks provide services to the user applications, we would like to examine elements of the system through an example.

3.2 Example 1: Cloud storage

Consider a problem of storing a data stream located on the server to the cloud storage located in data center. We measured time required to perform such transfer using a cluster of machines with software running aforementioned storage systems.

The storing operation was performed several times in experiments using different environments running on five different types of software and the amount of transferred data ranging from 1 Mb to 340 Mb. Set of experiments translated into a projection of the dependency between the size of data and the transfer time, which is shown on the graph in Figure 5.

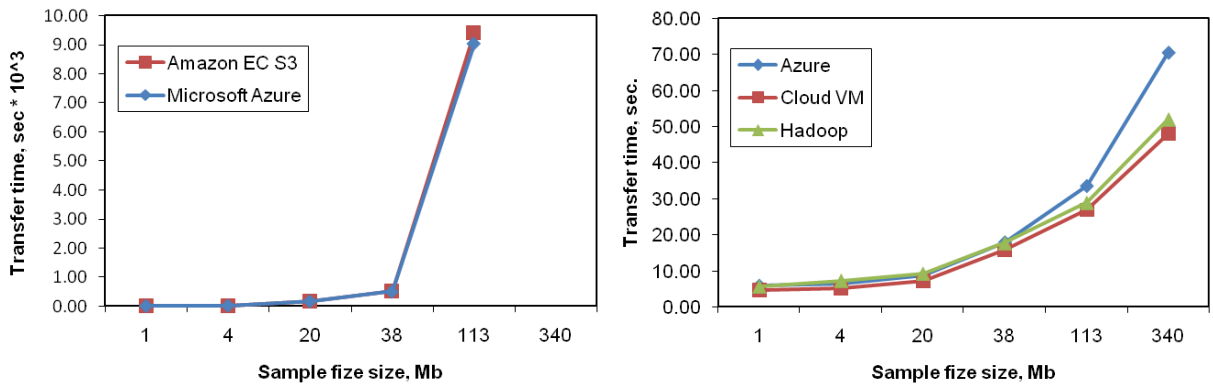

Figure 5 Sample file transfer to the cloud storage

The obtained timings on the graph indicate that operations on the cloud are significantly slower than those which are based on the internal cloud storage which is apparently because the storage is located much closer to the originator.

3.3 Example 2: Cloud job executing

Another problem which was evaluated is a problem of calculating the prime numbers with the specified number of digits. This problem is known as one which has asymptotic complexity of logarithmic function [11]. We evaluated time required to perform such calculation using a cluster of machines with software running aforementioned job scheduling systems and compared the obtained results as shown in Figure 6.

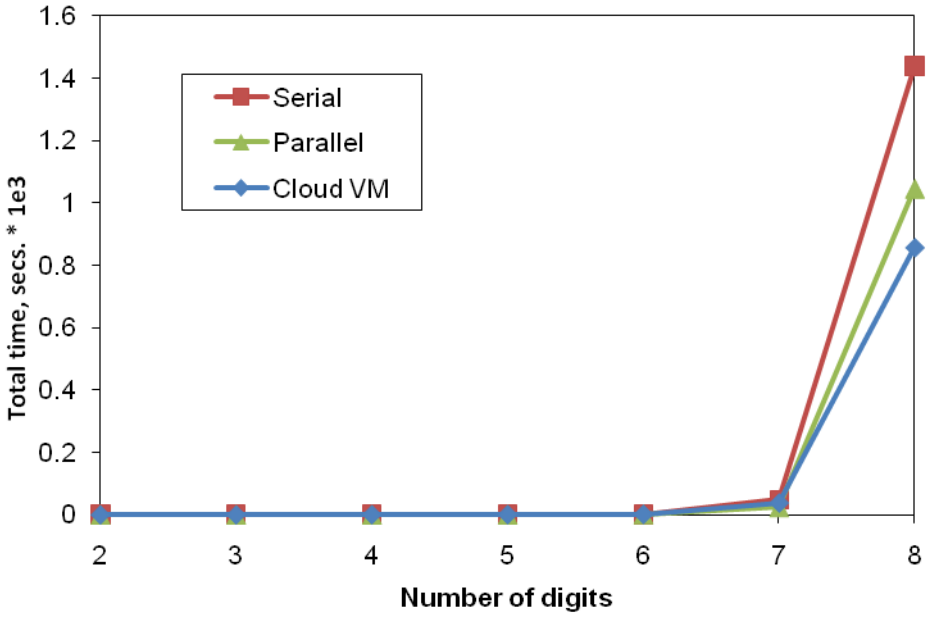

Figure 6 Computing prime numbers

The amount of time required to calculate prime numbers with specified number of digits grows drastically once the boundary reaches millions. Given problem can be calculated with us-

ing serial algorithm based on the Sieve of Eratosthenes [12], which is the first curve on the graph. Next, a parallel implementation of the algorithm was executed using the parallel algorithm where the original task is decomposed into features which return result and, finally, total count and the prime numbers itself are aggregated into resulting array. It is worthwhile to note that the parallel implementation is capable of executing features without prioritization. This capability is desired even for this simple problem because the greater the numbers, the more computational cycles are required to determine a prime number. The proposed Cloud-based Job scheduling system performs calculations of the features while it might perform a little slower with small numbers because of the additional overhead incurred due to synchronization between cluster nodes. Nevertheless, as a number of digits becomes greater, the total calculating time is less than that for other methods. These results are coherent with other problems we have evaluated.

4 Discussion of results

We have presented the major blocks which are sought to lay out a good foundation for the cloud framework with distinctive approach to persistent storage, temporal data, and job execution. We have shown how to evaluate the obtained results when the performance of operations is critical and reliability and security are on pinnacle of the requirements. This, we feel, raises a strong argument for keeping data close to the originator and clearly distinct internal cloud operations from similar operations that form a global cloud of the application's logic. This ensures that the implementation meets those obligations which are usually imposed in the corporate world. We yet to investigate the hybrid approach where depending on the business rules data may be stored either in an internal cloud or a global cloud when performance depends on the ratio of the local vs. global cloud operations. This does raise the question of whether there are any other kinds of invariants that would be useful but which cannot be expressed in our system. We have concluded that further design that increases the effectiveness of the framework such as creating open specifications and standards for implementations of cloud providers are possible. The implementation in the form presented in this paper was targeting heterogeneous environments by using the portable high-level language with the idea in mind that it will implemented for major platforms, including JRE, .Net which is coherent with the goal for keeping interoperability among top priorities. Thus, we believe, it is easy to see how these ideas could be applied to an implementation of the cloud-based virtual machine.

5 Conclusion

The virtual machine development is illustrated from the theoretical characterization of cloud computing. The main advantage is to obtain a complete checking of results in emerging technology. The paper illustrates the use of generic development frameworks based on general virtual machine structures; the generically of proof-based development is a way to improve the existing frameworks. The fundamental question is to be able to state what the problem to solve is and what development environment and structures are required for a given problem. Future work will develop new instantiations for the cloud-based virtual machine.

References

1. Chappell D., A Short Introduction to Cloud Platforms (http://www.davidchappell.com/CloudPlatforms--Chappell.pdf), David Chappell & Associates, 2008.
2. The Khronos Group Releases OpenCL 1.0 Specification (http://www.khronos.org/registry/cl/specs/opencl-1.0.29.pdf), Khronos Group, 2008.
3. Volkov V., Kazian B., Fitting FFT onto the G80 Architecture, Project Report, University of California at Berkeley, 2008.
4. Dollimore J., Kindberg T., Coulouris G., Distributed Systems: Concepts and Design, Addison Wesley, 2005.
5. Li, K. Shared Virtual Memory on Loosely Coupled Multiprocessors. PhD thesis, Department of Computer Science, Yale University, September 1986.
6. Tanenbaum A., Modern Operating Systems (Third Edition), Prentice Hall, 2007.
7. Madon D., Sanchez E. Monnier S. A Study of a Simultaneous Multithreaded Processor Implementation. Europar '99. August 1999.
8. Zhichun Zhu, Zhao Zhang A Performance Comparison of DRAM Memory System Optimizations for SMT Processors by and. HPCA-11. February 2005.
9. Robert Rinker, Roopesh Tamma and Walid Najjar. Evaluation of Cache Assisted Multithreaded Architecture MTEAC-1. January 1998.
10. Alex Settle, Joshua Kihm and Andrew Janiszewski, Architectural Support for Enhanced SMT Job Scheduling. PACT 2004. September 2004.
11. Bernstein, D. J. "An Exposition of the Agrawal-Kayal-Saxena Primality-Proving Theorem." 2002.
12. O'Neill, Melissa E., "The Genuine Sieve of Eratosthenes", Journal of Functional Programming, Cambridge University Press, 2008.