

An Intalio White Paper

Ismael Chang Ghalimi, CEO — May 2010

Cloud Computing is Memory Bound

Cloud Computing is all about elasticity be it for CPU, memory, or storage. But some elements are easier to make elastic than others, and experience shows that memory is by far the most challenging of them. For this reason, Cloud Computing should be considered memory bound.

Introduction

One of the main benefits of Cloud Computing is elasticity¹. Public Clouds Services such as Amazon Web Services provide services for elastic compute (EC2), elastic storage (S3), and many other core infrastructure components. One service is remarkably absent though: elastic memory. The reason for this is fairly simple: building a generic service for elastic memory is virtually impossible based on technology available today.

The Elastic Memory Challenge

Elastic services for Cloud Computing all rely on some form of distribution of jobs and pooling of resources. For example, the popular Amazon Elastic MapReduce² lets users process vast amount of data by distributing computation on a large pool of servers using the Amazon Elastic Compute Cloud (Amazon EC2) and distributing data on a large pool of drives using Amazon Simple Storage Service (Amazon S3). Such an architecture works for the MapReduce software framework³, because such a framework was explicitly designed to support mass scale distribution, whereby jobs can be easily partitioned into smaller tasks that can be distributed across many servers that have enough on-board memory to perform their respective tasks, without having to communicate with each other too frequently. In other words, the map and reduce functions used in functional programming are fairly easy to parallelize. Unfortunately, this is not the case for most functions used in enterprise IT.

Applications such as database transaction processing require large amounts of memory, and mandate that such memory be located close to the CPUs responsible for processing the transactions. This is the reason why large database applications have traditionally been deployed on mainframe computers or large servers using architectures like Symmetric Multiprocessing (SMP) or Non-Uniform Memory Access (NUMA) to pool memory across multiple CPUs through dedicated buses or crossbar switches. While these systems scale fairly well up to a few dozen CPUs, their cost per GHz of CPU and GB of memory is about 2.5 times the cost of standard servers. Furthermore, beyond 64 or 128 CPUs, adding more CPUs to these systems actually start to degrade their performance. Essentially, they're only scalable up to a certain point, and cannot be considered as elastic.

Over the years, more cost effective alternatives have been developed in order to leverage off-the-shelf servers. Some like ScaleMP⁴ work at the firmware level, others like RNA⁵ work at the kernel level, and yet others like GigaSpaces⁶ or Terracota⁷ work at the application server level. Nevertheless, they either meet the same scalability limits as SMP or NUMA systems (ScaleMP, RNA), or require applications to be developed for them specifically (GigaSpaces, Terracota), and none can make a database server elastic.

Memory is the Bottleneck

While creating an elastic memory infrastructure is not an easy task, elastic compute and elastic storage are quickly becoming mainstream. These technologies were originally developed in computer science labs during the 90's, then used for large-scale commercial applications by Internet search engines such as AltaVista and Google a short decade later. Eventually, they were popularized by Amazon Web Services with EC2 and S3, and then made available to corporate data-centers with open source projects such as Eucalyptus (elastic compute), Gluster (elastic storage), or Hadoop (elastic MapReduce). Ultimately, they will become part of any operating system, and the problem of elastic compute and storage will be considered solved.

All the while, little progress is being made to make memory elastic, thereby creating a critical bottleneck in any Cloud Computing architecture. And as it turns out, the Cloud Computing architecture is making matters even worse: not only does Cloud Computing requires a lot of memory, it does require more than any other architecture. For example, with the client-server model⁸, memory was fairly well balanced between SMP servers and fat clients. But with the browser-based thin client architecture privileged by the Cloud Computing model, things got totally off balance, and memory requirements have shifted toward the server (or the Cloud). The adoption of the AJAX model⁹ helped a little bit compared to the plain HTML model of the late 90's, but not enough to solve the elastic memory problem.

CPU Virtualization

First and foremost, Cloud Computing requires massive amounts of memory when CPU Virtualization is used for the purpose of multi-tenancy. While other virtualization techniques like JVM Virtualization¹⁰ or Application Server Virtualization can reduce the amount of memory that must allocated to every tenant (Cf. On Multi-Tenancy¹¹), CPU virtualization remains the most popular today. CPU Virtualization requires so much memory, because it relies on the allocation of a complete stack – operating system, application server, database server, application stack – for each and every tenant.

Memory requirements largely depend on which applications are being used, but as a point of reference, an aggressively optimized stack designed for Java applications requires at least 250MB per tenant, and 500MB is a more realistic target. Technologies such as OSGI¹² can help reduce memory requirements, but only incrementally. At the time of writing, a Public Cloud Service such as Salesforce.com had 72,500 customers (tenants) and 2,100,000 subscribers (users), which amounts to 29 users per tenant¹³. Assuming an average memory allocation of 25MB per user on top of a base memory allocation of 500MB per tenant, each tenant would require about 1.25GB, and a Salesforce.com class Cloud would need about 88.75TB of memory.

Database Elasticity

Immediately following CPU Virtualization, the need for memory elasticity comes from transactional database applications. In a Cloud Computing environment, making the database elastic is a major challenge, which is the reason why Cloud Services based on memory virtualization like Salesforce.com enforce so many execution governors and limits¹⁴. While non-relational datastores (aka NoSQL¹⁵) are gaining in popularity, most enterprise applications require a relational database, which is inherently difficult to distribute across multiple hosts. And while single-master clustering can help in cases where many more READ transactions than WRITE transactions are performed by the application (Cf. Elastic Database¹⁶), multi-master clustering does not scale beyond a few master servers. As a result, master database servers have to be hosted on systems that provide as much on-board memory as possible. Fortunately, the chipsets supporting modern multi-core CPUs are designed to accommodate larger and larger amounts of memory. For example, Intel's Xeon X7560 supports up to 1TB of memory.

Interestingly, most relational database servers currently in use handle less than 1TB of data. Even though no reliable statistics on this topic are publicly available, it is very likely that less than 1,000 transactional databases of more than 1TB in size are deployed in the entire world today. Furthermore, the largest transactional databases currently in use today, many of which track phone calls, grow in size significantly slower than the maximum amount of on-board memory that can be handled by off-the-shelf x86 servers, which seems to be doubling year over year. Keeping these facts in mind, the vast majority of transactional databases can fit in the on-board memory of a single server today, making the need for memory sharing across multiple servers essentially non-existent, at least as far as database applications are concerned. And such a server is actually more affordable than most people think. At the time of writing, one can buy a four-socket 4U rack server with 1TB of DDR3 memory from Dell (Dell PowerEdge R910) for less than \$85,000. Two years ago, it would have cost four times as much.

What this means ultimately is that a well-designed Infrastructure as a Service based on server clustering for the application tier, single-master clustering for the database tier, and off-the-shelf servers with large amounts of on-board memory for the underlying hardware will be able to handle virtually any database applications, from the smallest to the largest, in an elastic-enough manner. And for the very few applications which database could not fit into the memory of a single server, two options remain: either dedicating a NUMA server (expensive from a hardware standpoint, but fairly straightforward), or migrating to a NoSQL architecture (preferred for many new Web-based applications that perform few mission-critical transactions).

Web Office

Another application requiring significant amounts of memory is Web Productivity, implemented by services such as Google Docs¹⁷ or Intalio|Web Office¹⁸. Both feature similar architectures based on the use of a headless version of OpenOffice.org deployed on the server and a lightweight AJAX client running on the Web browser (Cf. Intalio|Web Office Architecture). Based on this architecture, every concurrent user requires a dedicated OpenOffice.org instance running on the server side. While memory requirements depend on which application is used and how large user-created documents are, a typical scenario will require around 250MB per user. As a result, a single server with 1TB of on-board memory will be able to handle around 4,000 concurrent users. Fortunately, this is a perfectly elastic application that does not require servers to share their memory in order to accommodate large numbers of concurrent users.

Memory Virtualization

While different techniques can be used to create an elastic-enough memory infrastructure for Cloud Computing as described above, another requirement emerges: making the best possible use of all available memory.

If a server with 1TB of on-board memory "only" costs \$85,000, 85% of the cost comes from memory itself. When adding the cost of networking and storage, it is fairly typical to find out that memory still accounts for over 50% of the total hardware costs. Why this matters is because ultimately, any waste of memory will translate into a waste of data-center resources, which include not only hardware, but also real estate and electricity for power and cooling. As a rule of thumb, data-center operation costs are roughly split in half between hardware amortization and utility costs (when taking manpower costs aside).

As a result, a waste of 10% of available memory will translate into a waste of 5% of available hardware. But since most Cloud applications are memory bound – meaning that memory is the bottleneck – memory waste spills over the entire hardware infrastructure. In other words, wasting 10% of all available memory leads to wasting the same 10% of all available hardware. And because the amount of electricity required to power and cool down a server is exactly the same whether memory is properly used (allocated) or not, the total waste translates into 10% of the overall data-center resources, which ends up costing 4 times as much as what the wasted memory initially cost. To make a long story short, wasting memory is a really bad idea.

This is where memory virtualization comes into play, not so much for addressing the need for elastic memory, but for reducing the amount of memory waste. The idea is pretty simple: instead of allocating a fixed amount of memory for each application as most hypervisors and Java Virtual Machines do today, all available memory can be shared across all running applications and allocated on demand. Unfortunately, such dynamic memory allocation is not possible with hypervisors today, which suggests that other means of virtualization should be used for the purpose of multi-tenancy when memory waste becomes too significant for it to be simply ignored (Cf. On Multi-Tenancy¹⁹). One of them is JVM Virtualization. And as it turns out, there are many reasons why this approach should be preferred over any other.

The idea behind JVM Virtualization is to provide a virtual JVM for each application (or tenant). This technology is currently offered by Azul Systems²⁰ and comes with a fantastic side benefit: it allows multiple Java applications to share the same memory space, in a secure manner, and without having to define a static heap size²¹ for each application. As a result, two birds get killed with one stone: multi-tenancy (with secure virtual JVMs) and memory waste reduction (with memory sharing).

This is particularly significant when considering the memory usage profile of most Web-based Java applications: at start time, they tend to use a very small amount of memory. But as more concurrent users start logging in, more objects must be allocated, and more memory gets used. Unfortunately, usage patterns tend to be hard to predict, therefore systems administrators have the tendency to over-allocate memory for these applications, thereby creating massive wastes of memory. It is estimated that on a time-adjusted basis, anywhere between 50% to 90% of all available memory is wasted in such a fashion. Coming back to our previous discussion, this translates into wasting 50% to 90% of all data-center resources...

Moving Forward

As we have seen, Cloud Computing is memory bound, and while elastic compute and elastic storage have almost become mainstream technologies, truly elastic memory is akin to science fiction today. That being said, a pragmatic review of the most relevant use cases leads one to devise architectures that can work around this problem, making a properly-designed Infrastructure as a Service (IaaS) elastic enough from a pure memory standpoint. Furthermore, emerging technologies such as JVM Virtualization will continue to improve the way memory gets used by Web-based applications, therefore leading to increased economies of scale when deploying large Cloud Computing platforms.

-
- ¹ <http://helium.intalio.com/benefits-of-cloud-computing>
 - ² <http://aws.amazon.com/elasticmapreduce/>
 - ³ <http://en.wikipedia.org/wiki/MapReduce>
 - ⁴ <http://www.scalemp.com/>
 - ⁵ <http://www.rnanetworks.com/>
 - ⁶ <http://www.gigaspace.com/>
 - ⁷ <http://www.terracotta.org/>
 - ⁸ http://en.wikipedia.org/wiki/Client%E2%80%93server_model
 - ⁹ [http://en.wikipedia.org/wiki/Ajax_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming))
 - ¹⁰ <http://www.intalio.com/jvm-virtualization>
 - ¹¹ <http://www.intalio.com/on-multi-tenancy>
 - ¹² <http://www.intalio.com/osgi>
 - ¹³ <http://en.wikipedia.org/wiki/Salesforce.com>
 - ¹⁴ http://www.salesforce.com/us/developer/docs/apexcode/Content/apex_gov_limits.htm
 - ¹⁵ <http://en.wikipedia.org/wiki/NoSQL>
 - ¹⁶ <http://www.intalio.com/elastic-database/elasticity>
 - ¹⁷ <http://docs.google.com/>
 - ¹⁸ <http://www.intalio.com/web-office>
 - ¹⁹ <http://www.intalio.com/on-multi-tenancy>
 - ²⁰ <http://www.azulsystems.com/>
 - ²¹ http://wiki.answers.com/Q/What_is_heap_size_in_Java



Intalio, Inc.
World Headquarters
644 Emerson Street, Suite 200
Palo Alto, CA 94301
United States

Worldwide Inquiries:
Tel: +1 (650) 596-1800
Fax: +1 (650) 596-1801
info@intalio.com
www.intalio.com



Intalio is committed to developing practices and products that help protect the environment.

Copyright © 2010, Intalio and/or its affiliates. All rights reserved. This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Intalio is a registered trademark of Intalio, Inc. and/or its affiliates. Other names may be trademarks of their respective owners.