

Early Observations on the Performance of Windows Azure

Zach Hill, Jie Li, Ming Mao, Arkaitz Ruiz-Alvarez, and Marty Humphrey
Department of Computer Science
University of Virginia
Charlottesville, VA 22904

ABSTRACT

A significant open issue in cloud computing is performance. Few, if any, cloud providers or technologies offer quantitative performance guarantees. Regardless of the potential advantages of the cloud in comparison to enterprise-deployed applications, cloud infrastructures may ultimately fail if deployed applications cannot predictably meet behavioral requirements. In this paper, we present the results of comprehensive performance experiments we conducted on Windows Azure from October 2009 to February 2010. In general, we have observed good performance of the Windows Azure mechanisms, although the average 10 minute VM startup time and the worst-case 2x slowdown for SQL Azure in certain situations -relative to commodity hardware within the enterprise- must be accounted for in application design. In addition to a detailed performance evaluation of Windows Azure, we provide recommendations for potential users of Windows Azure based on these early observations. Although the discussion and analysis is tailored to scientific applications, the results are broadly applicable to the range of existing and future applications running in Windows Azure.

Categories and Subject Descriptors

H.3.4 [Information Storage and Retrieval]: Systems and Software – *distributed systems, performance evaluation (efficiency and effectiveness)*

General Terms

Measurement, Performance, Experimentation.

Keywords

Cloud computing, scalability, cloud storage services, virtual machines, Windows Azure.

1. INTRODUCTION

Cloud computing [2] has burst onto the high performance computing scene in recent years and has established itself as a viable alternative to customized HPC clusters for many users who do not have the resources -- either time or money -- to build, configure, and maintain a cluster of their own. The ability to pay only for resources that are utilized and the apparent ease by which

resources can be expanded, deployed, and removed from service are very attractive. Many eScience developers are increasingly looking to create data-intensive applications that have highly variable resource requirements over time and can take advantage of the pay-as-you-go cost model.

As more cloud providers and technologies enter the market, developers are faced with an increasingly difficult problem of comparing various offerings and deciding which vendor to choose for deploying an application. One critical step in the process of evaluating various cloud offerings can be to determine the performance of the services offered and how those match the requirements of the application. Even in situations where other factors ultimately dominate the choice regarding potential cloud platforms -e.g., cost per unit time of a virtual machine or application-hosting environment in the particular cloud-, it is important to consider performance ramifications of design decisions to ensure maximum value of cloud applications.

On February 1, 2010, Microsoft announced the commercial availability of their cloud offering, the Windows Azure Platform [13]. As evidenced by many of the presentations at Microsoft's Professional Developers Conference -PDC-, Microsoft has made a significant investment into Windows Azure to make it attractive to the Information Technology community. This investment is both the hardware/software necessary to run the Windows Azure cloud and also enhancements into the Microsoft software development tools -e.g., Visual Studio- to address the challenges of writing, deploying, and managing a cloud application. While the target audience of Windows Azure has understandably been the broad business community, it is worth noting that on February 4, 2010, Microsoft and the National Science Foundation -NSF- announced a partnership aimed at getting the science and engineering community using Windows Azure.

The purpose of this paper is to provide a quantitative analysis of the performance of the Windows Azure Platform. Through our existing collaborative partnership with Microsoft Research, we were able to evaluate the services at scales not readily available to users of the early Community Technology Preview -CTP- release--up to 192 concurrent instances. This partnership, however, does not imply interior knowledge of the Azure infrastructure, and throughout this paper our perspective is the same as other users of the Azure platform. Our methodology is to assume that Windows Azure has already been chosen as the target cloud for whatever reason, and that now the developer is facing the challenge of architecting his/her cloud application to accommodate performance considerations. In other words, to maintain a reasonable scope of this research effort -and this paper-, we do not directly compare performance of Windows Azure to other clouds. We plan to address this issue and provide this detailed report in the near future.

The Windows Azure Platform is composed of three services: Windows Azure, SQL Azure, and AppFabric. This paper focuses on Windows Azure, which encompasses both compute resources and scalable storage services, and SQL Azure, which provides traditional SQL Server capabilities for databases up to 10GB in size. We omit the Windows Azure platform AppFabric from discussion as it is a newly released feature, which we have not had time to evaluate properly.

Our evaluation of the Windows Azure service begins with the performance of its compute resources and its three primary storage services: Blobs, Queues, and Tables. Because these are the basic storage components that scalable Windows Azure applications are built upon, it is important to understand their performance characteristics as scale increases. We do not present an evaluation of Azure Drives, the NTFS storage abstraction because it was just recently released into beta testing in February 2010. We then evaluate virtual machine instantiation time because instance acquisition and release times are critical metrics when evaluating the performance of dynamic scalability for applications. We also present an evaluation of direct instance-to-instance TCP performance as this mechanism provides an alternative to the other storage services for communication between instances that has lower latency. For the SQL Azure service, we present the performance implications of running a database server in the cloud, the influence of client location, scalability in terms of the number of clients and availability over time. We use the TPC-E benchmark, which is a database benchmark defined by the Transaction Processing Council that represents the operations of an OLTP system.

In general, we have observed good performance of the Windows Azure mechanisms, although the average 10 minute VM startup time and the worst-case 2x slowdown for SQL Azure in certain situations -relative to commodity hardware within the enterprise- must be accounted for in application design.

Finally, we summarize our experimental data into several specific recommendations for developers using Windows Azure Platform. In these recommendations we address virtual machine instances, the storage services, SQL Azure and our experience in testing and developing cloud applications. Although the discussion and analysis is tailored to scientific applications, the results are broadly applicable to the range of existing and future applications running in Windows Azure.

The rest of this paper is organized as follows: Section 2 surveys related work. We start our analysis of the Windows Azure storage services in Section 3. We then proceed to discuss the results of our experiments with the Azure computing services in Section 4. Section 5 presents our results with SQL Azure. We discuss the implication of our results for both users and cloud providers in Section 6. Finally, we conclude with Section 7.

2. RELATED WORK

In the research community, there is an increasing recognition of performance concerns of clouds and their underlying technologies. For example, Menon et al. [12] and others [15] evaluated the performance overhead of Xen [3], a software virtualization technology which is a popular choice as the low level virtual machine manager by several cloud providers. Xen

has been shown to impose negligible overheads in both micro and macro benchmarks [17].

A higher level analysis is provided by Garfinkel [8], who evaluates some of the cloud services that Amazon provides. Our earlier work compared the performance of cloud platforms with local HPC clusters for scientific applications [10]. Another report examines the feasibility of using EC2 for HPC in comparison to clusters at NCSA [16]. This comparison pits EC2 against high-end clusters utilizing Infiniband interconnects.

There are also studies that focus on a specific scientific application, such as DZero [14] or Montage [5], to evaluate the possibility of migrating existing applications and data to the cloud, based on performance and cost parameters. Workflows [11] and service-oriented applications [6] have also been the object of study. Another study reports on the possibility of running coupled ocean-atmosphere simulations on EC2 [7].

The research reported in this article complements this earlier work by providing a direct measurement of the mechanisms and APIs of a specific cloud, Windows Azure.

3. AZURE STORAGE SERVICES

In this section we start our analysis with the performance tests of all three Azure Storage Services: blob, table and queue. For each service we measure maximum throughput in operations/sec or MB/sec and the scalability of the service as a function of the number of concurrent clients, among other service-related metrics. For all our tests we use from 1 to 192 concurrent clients.

In general, the scalability of these storage services may be a problem for large-scale applications with multiple concurrent clients if the data is not adequately partitioned or replicated - all of the storage services are automatically and transparently triple-replicated. The performance penalty caused by increased concurrency must be taken into account in order to meet the application's requirements, and we provide several data points to help the software developer make decisions about the application's architecture and scale.

3.1 Blob

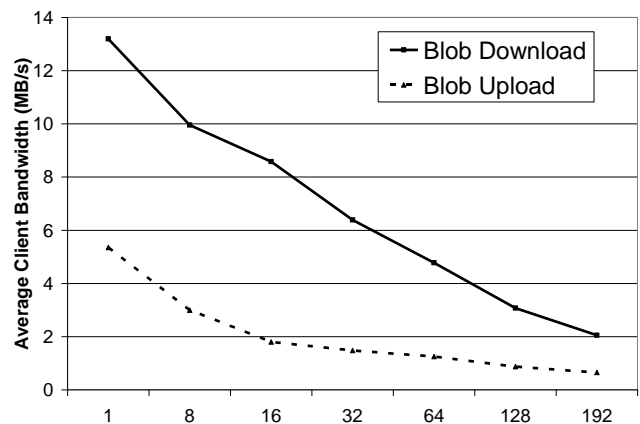


Figure 1: Average client blob download bandwidth as a function of the number of concurrent clients

The Blob service in Windows Azure establishes a storage hierarchy: storage accounts have multiple containers, which hold

one or more blobs. Each individual blob can store one terabyte of data and some associated metadata. In this section we analyze the performance of both the download and upload of data onto blobs.

For our blob download test we use a single 1GB blob that is stored in Azure. Then, we start a number of worker roles -from 1 to 192- that download the same 1GB blob simultaneously from the blob storage to their local storage. For the upload test, since the different worker roles can't upload the data to the same blob in Azure storage, a different test is used: the worker roles will upload the same 1GB data to the same container in the blob storage, using different blob name. We run the same test three times each day. Although we have collected data for several days at different times, the variation in performance is small and the average bandwidth is quite stable across different times during the day, or across different days.

The maximum server throughput for the blob download operation was 393.4 MB/s, which was achieved by using 128 clients. For the blob upload operation, the maximum throughput was 124.25 MB/sec, which was observed in our experiments with 192 concurrent clients.

Figure 1 shows the average client bandwidth as a function of the number of concurrent clients attempting to download the same blob. The bandwidth for 32 concurrent clients is half of the bandwidth that a single client achieves. Using more concurrent clients -up to 128- increases the total aggregate bandwidth, although this comes with the price of much slower clients.

Figure 1 also shows the performance of the upload blob operation. The scalability test results for the upload operation show similar results to the download ones, as we can see the similarities between both curves in the graph. However, because we are now only uploading the 1GB file to the same container as different blobs, we are not accessing the same blob object as in the download operation. We conclude that the operations "uploading blobs to the same container" and "downloading the same blob" both suffer from the same concurrency constraints.

The overall blob upload speed -including the baseline- is much slower than the download speed. For example, average upload speed is only ~0.65 MB/s for 192 VMs, and ~1.25MB/s for 64 VMs. This lower speed may be linked to the internal network upload policies, but also to constraints on write operations. The upload speed may be limited by the complexities involved in infrastructure-level operations for creating new blob objects, such as record generations, replication operations, etc.

3.2 Table

A table in Azure is a set of entities with properties, where each property can have various types and the table has no defined schema. In this sense, it is very different from a table in a relational database. A single table has the capability to store a large number -billions- of entities holding terabytes of data. We have examined the performance of 4 operations from the Azure Table API: Insert, Query, Update and Delete. We have run our experiments with different entity sizes: 1 KB, 4 KB, 16 KB and 64 KB. We used from 1 to 192 concurrent clients to study the scalability of each type of table operation. We have found that the shape of the performance curves for different entity sizes are similar, except for some exceptions which are noted below.

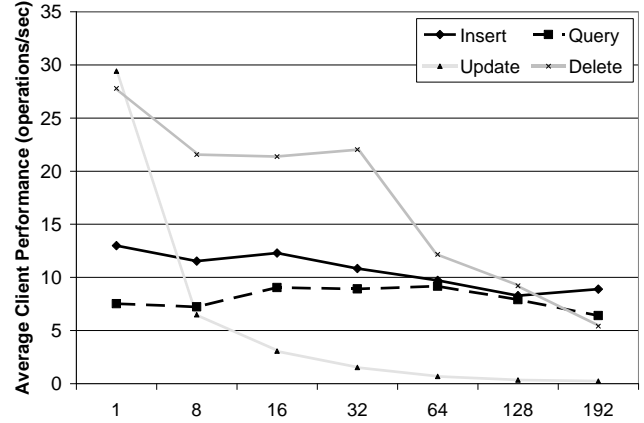


Figure 2: Average client Table performance as a function of the number of concurrent clients. Entity size is 4 KB.

For each test case, our experiments performed the following steps: we start with the Insert experiment using different number of concurrent clients, where each client inserts 500 new entities into the same table partition; after the Insert experiment, the table partition includes ~220K same-size entities. Next, we perform Query operations over the same partition by using a partition key and row key, and each client queries the same entity 500 times. Since currently Azure tables are indexed on PartitionKey and RowKey only, this key-based query is the fastest query option. Azure table also support query on table properties other than the keys, but we didn't evaluate their performance here. For the Update experiment, each concurrent client updates the same entity in the partition and repeats the operation for 100 times. Here we only tested with the *unconditional updates* option as it doesn't enforce atomicity of each update request, so that different clients can issue update requests to the same table entity at the same time. Finally, in the Delete experiment, each client removes the same 500 entities it inserted in the first step of our experiment.

The result of our experiments is summarized in Figure 2 and Table 1. Figure 2 presents the data from the point of view of the client, that is, how many operations per second can concurrent clients sustain? For both Insert and Query, the performance of the clients decreases as we increase the level of concurrency. However, we think that even with 192 concurrent clients we have not hit the maximum server throughput for these two operations. The Update and Delete tests show more drastic performance declines as we increase the number of clients, though. These two operations has high initial throughput with only 1 client, but then slow down drastically as the number of concurrent clients increases. The maximum throughput for these two services is reached at 8 concurrent clients for the Update operation and 128 for the Delete operation.

Our experiments show that the performance curves for other entity sizes are similar as Figure 2, except for the following exceptions during the Insert and Delete tests: For the Insert test on 64 KB entities with 192 concurrent clients, only 89 clients successfully finished all 500 insert operations, and the other 103 client have encountered timeout exceptions from the server. With 128 concurrent clients inserting 64KB entities, only 94 clients successfully finished all 500 operations. This indicates that we may hit the table service capability limit on the above two

combinations with large entity size and high concurrency. We have also observed similar behaviors during the Delete tests.

Table 1 summarizes the observed maximum throughput for the Table service for different entity sizes. The number in the parentheses indicates the number of concurrent clients at which point the maximum throughput is achieved. As we can see, Insert is the most sensitive operation to the size of entities, whose effect can be as high as 13 times less operations per second. On the other hand, the throughput of Query operations is the least sensitive to different entity sizes.

Table 1: Maximum observed Table throughput in ops/sec as a function of entity size. The numbers in parentheses are the number of concurrent clients

Operation	1 KB	4 KB	16 KB	64 KB
Insert	2237 (192)	1706 (192)	520 (128)	165 (128)
Query	1525 (192)	1011 (192)	1153 (192)	1139 (192)
Update	85 (16)	52 (8)	59 (8)	37 (16)
Delete	2227 (192)	1178 (128)	1281 (192)	622 (64)

Finally, we have run experiments that compare the performance of the Table and Blob services. That is, for storing information that does not need to be queried –so it could be stored as a blob– and that does not exceed the table entity size limit –so it could be stored in a table too–, we want to find out what is the best option for the programmer. Table 2 answers this question. We run the same experiment: 1, 8 and 16 concurrent clients performing insertions of objects from 1 KB to 64 KB. We have used both a blob container and a table as the backend storage system. The results show that Blob is preferable if the amount of data to insert is between 4 KB and 64 KB. The Blob performance can be as low as a fourth of the Table performance for small objects of 1 KB: it takes around 200 ms to create a new blob of 1 KB versus 40 to 70 ms to insert a 1 KB table entity. Depending on the number of concurrent clients and the size of the data objects –if 4 KB or greater– to be stored the performance of Blob is between 35% to 3 times faster than Table. Surprisingly, it takes between 40 to 60 ms to insert a blob of 4 KB, which means that it is 5 times faster to create a 4 KB blob than a 1 KB blob. Due to the black box nature of the cloud services we do not have an explanation for this behavior.

Table 2: Average client Blob insert performance in ops/sec compared to Table insert performance

# Clients	1 KB	4 KB	16 KB	64 KB
1	0.22x	1.35x	1.43x	1.44x
8	0.29x	2.05x	2.86x	2.60x
16	0.36x	2.13x	2.56x	2.79x

3.3 Queue

The main purpose of Queues in Windows Azure is to provide a communication facility between web roles and worker roles. For our queue test we use one queue that is shared among several worker roles –from 1 to 192–. We examine the scalability of three Queue operations: Add, Peek and Receive. For each operation we

run the test with different message sizes: 512 bytes, 1 KB, 4 KB and 8 KB. As it was the case with Table, the shape of the performance curves for each message size is similar and we choose to show the results for 512 bytes.

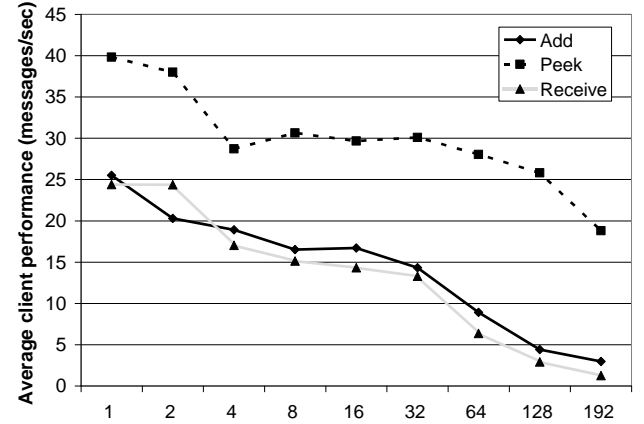


Figure 3: Average Queue client performance as a function of the number of concurrent clients. Message size is 512 bytes.

Figure 3 shows our results. In general, the operations Add and Receive display similar trends. Peek message is the fastest operation, since it does not need to alter the visibility of the message. Thus, the different replicas of the queue do not need to be synchronized on a Peek operation and all of these requests can be run in parallel. Add and Receive require a different implementation, though. Both of these operations require synchronization among distributed objects: the Add operation needs to add the message to the same places in each replica of the queue; the Receive operation needs to assign the triple-replicated message to only one client.

Table 3 presents the maximum throughput of each operation based on the size of the message. For the Add and Receive operations, the maximum performance peaks at 64 concurrent clients. For Peek, however, we present the throughput that was achieved with 192 concurrent clients. We believe that we have not exercised this operation on the Queue service to the maximum of its capacity, albeit the throughput increase from 128 instances to 192 instances starts showing diminishing returns. Limitations on the number of virtual machines that we can start on Windows Azure prevent us from running this experiment at a higher scale.

Table 3: Maximum observed Queue throughput in ops/sec as a function of on message size. The numbers in parentheses are the number of concurrent clients

Operation	512 B	1 KB	4 KB	8 KB
Add	569 (64)	565 (64)	556 (128)	502 (64)
Peek	3878 (192)	3609 (192)	3388 (192)	3369 (192)
Receive	424 (64)	407 (64)	396 (64)	393 (64)

We have also run some experiments to test the influence of the size of the queue on the performance of the mentioned operations. We have found that there is not much variation at all as the queue grows in size from 200 thousand to 2 million messages.

4. AZURE COMPUTING SERVICES

In this section, we discuss computing instance acquisition and release time in Windows Azure and TCP communication between different virtual machines instances.

4.1 Dynamic Scalability

We believe that computing instance acquisition time is a critical metric to evaluate the efficiency of dynamic scalability for cloud applications. We have written a test program that uses the Windows Azure management API to collect timing information about each possible action on Azure virtual machine instances. We manage two types of Virtual Machines: web roles and worker roles. In addition, Azure offers four types of VM size: small, medium, large and extra large. By combining these two parameters for each test case we create a new Azure cloud deployment.

For every run of our test program, it randomly picks a role type and a VM size, and creates a new deployment. We choose the number of instances in each deployment based on the VM size: 4 instances for small, 2 for medium and one for large and extra large. Then our test program measures the time spent in all five phases – create, run, add, suspend and delete. These phases are divided based on Azure deployment and instance status.

1. Create: in this phase, we record the wall clock time from application deploy request initiation to the time when Azure indicates the deployment is ready to use. In our test, all the deployment packages are stored in Azure blob storage services.
2. Run: when the deployment is successful, the test program initiates a “Run” request to start the VM instances in the deployment. We measure the time from the start of the request to the time when all VM instances are ready to use – status changes from “stopped” to “ready”.
3. Add: after the instances are started running, the test program initiates a “Change” request and doubles the number of running instances. We measure the time that takes these newly added instances to become ready –status changes to “ready”.
4. Suspend: when all the VM instances are running, we suspend all the running instances in the deployment and measure the time spent to terminate each Azure VM instance - status changes from “ready” to “stopped”.
5. Delete: After all the running instances are suspended, our test program initiates a “Delete” request and removes the current deployment.

Table 4: Worker role VM request time in seconds

Size	Statistic	Create	Run	Add	Suspend	Delete
Small	AVG	86	533	1026	40	6
	STD	27	36	355	30	5
Medium	AVG	61	591	740	37	5
	STD	10	42	176	12	3
Large	AVG	54	660	774	35	6
	STD	11	91	137	8	6
Extra	AVG	51	790	N/A	42	6
Large	STD	9	30	N/A	19	5

Table 5: Web role VM request time in seconds

Size	Statistic	Create	Run	Add	Suspend	Delete
Small	AVG	86	594	1132	86	6
	STD	17	32	478	14	2
Medium	AVG	61	637	789	92	6
	STD	10	77	181	17	6
Large	AVG	52	679	670	94	5
	STD	9	40	155	14	3
Extra	AVG	55	827	N/A	96	6
Large	STD	16	40	N/A	3	8

From Dec 17, 2009 to Jan 09 2010, we collected data from 431 successful runs. The VM startup failure rate, taking into account all of our test cases, is 2.6%. Starting from Jan 1st 2010, Windows Azure changed from CTP to commercial platform. Our observations did not find clear performance differences between these two periods. The output data is shown in Table 4 and Table 5. From these tables we draw the following observations:

1. Web role VM instances need longer time to startup than worker role instances. For all VM sizes, web role takes 20 ~ 60 seconds longer than Work role VMs. Such observation is consistent with our expectation, since each web role requires a more complex initialization than a worker role -e.g. IIS support and Azure load balancer registration. Also, large VMs take longer time to startup than small VMs.
2. The average time to start a worker role small instance is around 9 min., while the average time to start a web role instance is around 10 min. The quickest time we observe is 7.5 min. for worker role and 9 min. for web role. For 85% of our test runs, the first small worker role instance becomes ready within 9 min. and for 95%, the first small worker role instance becomes ready within 10 min. For 80%, the first small web role instance becomes ready within 10 min and for 90%, the first small web role instance becomes ready within 11 min.
3. Azure does not serve a request for multiple VMs at the same time. That is, there is a lag between the time the first instance becomes available and the following ones. For both worker role and web role small instances, we have observed a 4 min. lag between the 1st instance and the 4th instance of our deployment.
4. Adding more instances to existing deployment takes much longer than requesting the same number of instances at the beginning.
5. Application deployment performance -create phase- is largely a function of the application size. A 1.2MB application starts 30 seconds faster than a 5 MB application. Note that our test deployment is stored in Azure storage service. If the application package is stored locally, the deployment time could take much longer because of the local network bandwidth limit.
6. Azure shows consistent performance for deployment deletion, around 6 seconds for all test cases.

4.2 TCP Communication

Windows Azure allows the programmer to define TCP or HTTP internal endpoints for the virtual machine instances in the deployment. This type of communication is highly coupled, works only in a point to point fashion and the application needs to define the protocol. However, it is a good complement to the Queue -low

coupling, multiple readers/writers, defined API- since these internal ports allow the VMs to talk directly with each other using a low-latency, high-bandwidth TCP/IP port. Therefore, we have measured the performance of this feature of Azure VMs based on three metrics: latency, bandwidth and bandwidth variability over time.

In our first experimental setup, we create a deployment with 20 small VMs. 10 of these VMs measure latency, and the rest measure bandwidth. Each virtual machine is paired with another one; each pair contains one server and one client. In order to measure the latency, the client measures the roundtrip time of 1 byte of information sent on the TCP channel, after communication has been established. For the bandwidth measurement the client sends 2 GB of information to the server –each run of this bandwidth test usually takes around 30 seconds.

Figure 4 and Figure 5 presents our results. We have collected for these two graphs a total of 10,000 measurements. Both figures present the histogram of our samples. Figure 4 shows that approximately 50% of the time the latency is equal to 1 ms; 75% of the time the latency is 2 ms or better. In general, the most common case is to find in the datacenter latency that is similar to our LAN. Figure 5 summarizes the bandwidth measurements. 50% of the time we find the bandwidth to be 90 MB/s or better. We assume that the physical hardware is Gigabit Ethernet, which has a limit of 125 MB/s. So in our experiments we have seen that is rather common for the VMs to have good bandwidth. However, for the lower end of the sample -15%- the performance drops to 30 MB/s or worse.

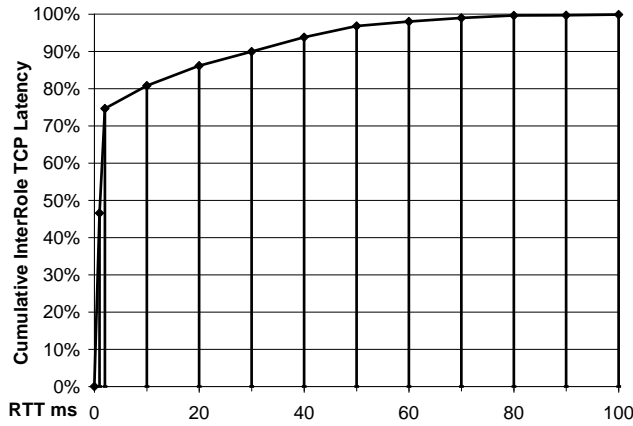


Figure 4: Cumulative TCP latency –RTT ms- between two small VMs communicating through TCP internal endpoints.

In our second experimental setup, we start a deployment with two VMs, one client and one server. These two VMs transfer the same 2 GB of data, but they do that at regular intervals -every half hour- for several days. We have plotted in Figure 6 our bandwidth measurements in MB/s. For the most part the bandwidth exceeds 80 MB/s, but there are times in which it can be as low as 10MB/s. Since we are using small instances, the resource manager will most likely allocate other instances from other deployments in the same physical hardware. We believe that the activity of these other neighboring instances can greatly influence the bandwidth observed by the application. Although the common case is still similar to our LAN, developers are warned that the variability is

high. TCP is the feature in which we have observed the highest degree of variability for the Windows Azure platform.

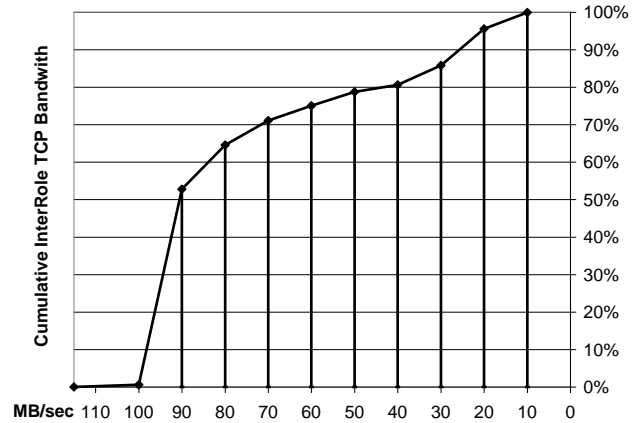


Figure 5: Cumulative TCP bandwidth –MB/sec- between two small VMs sending 2 GB of data through TCP internal endpoints.

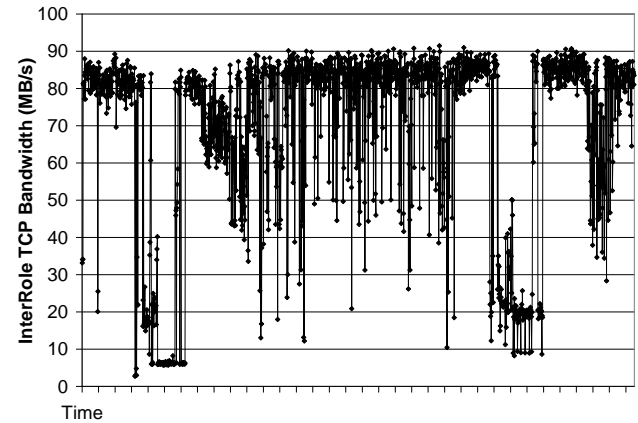


Figure 6: Variation of the InterRole TCP Bandwidth –MB/sec- between two small VMs.

5. SQL AZURE DATABASE

After an initial attempt to provide new data services models designed to be scalable, but with limited functionality, the Azure cloud platform started offering a full SQL product as an integral part of their service. The widespread use of the relational model in all kind of applications makes the SQL Azure Database a crucial part of the cloud infrastructure. Essentially, this product is a modified version of SQL Server that runs on the Azure platform and is compatible with Microsoft's database protocol Transact-SQL.

Similarly to other Azure products, SQL Azure focuses on high availability, scalability, ease of deployment and automatic management. In this section, we analyze these aspects together with the performance of SQL Azure to offer a complete picture of what this platform offers to new developers. Our test application is the TPC-E benchmark, whose specification is published by the Transaction Processing Council. The TPC-E benchmark simulates an OLTP workload, and is designed to replace the old TPC-C benchmark.

Our experimental setup is the following one: in the Azure cloud we have deployed a SQL Azure server with our 3 GB TPC-E database and we deploy up to 8 extra large instances -with eight cores and 8GB or RAM each- to function as TPC-E clients. Locally, we have a Windows Server 2008 in a quad core with 8 GB of RAM running SQL Server 2008, and 3 TPC-E client machines -dual cores- connected by a LAN.

5.1 Single Thread Client Performance

Graphs in Figure 7 show the performance comparison of the different SQL servers with clients that run in the same machine - Local client to local server-, across a LAN -Local client to server in LAN- and across the datacenter -Azure client to Azure server-. The TPC-E client runs a single thread of execution for four hours, including a ramp-up period of 30 minutes. The TPC-E benchmark consists of 10 different transactions, each of them consists of different SQL queries that include selections, updates, inserts and deletes across the 33 tables that compose the TPC-E database. We measure the time it takes to complete the transaction from the client side and show the average of each type of transaction -including a 95% confidence interval on the population mean as the error bar- and the general average, labeled as TPC-E.

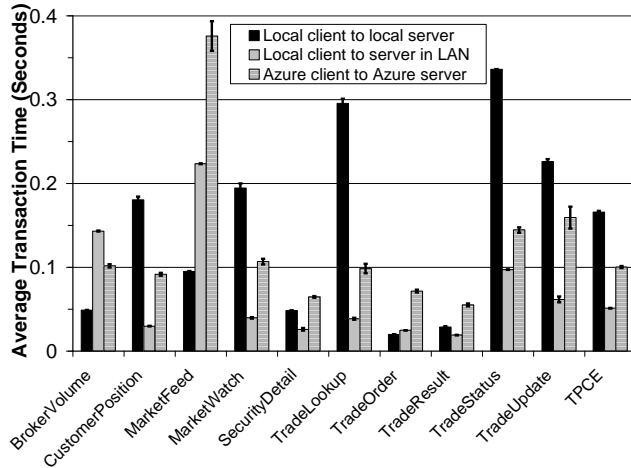


Figure 7: TPC-E transaction times in seconds for each client and server location.

There is, on average, almost a 2x difference in speed when we compare an Azure VM client querying a SQL Azure database with a local client querying a SQL Server across our LAN. The slowdown for “Local client to local server” is caused by SQL Server taking too many resources, which slows down the client. If the user wishes to locate the Azure VM client outside the cloud - mobile clients, saving costs, collaboration across multiple sites, etc. - we have observed that there is an order of magnitude of slowdown -not included in this graph-. This additional overhead is caused by latency of communications between our client in Virginia and the Azure server in the Southern United States datacenter.

This graph shows that, based on performance, a local deployment is preferable than a purely cloud deployment. In the next sections, however, we will discuss other aspects that developers should take into account.

5.2 Scalability

In order to evaluate the scalability of both local and Azure SQL servers, we start several TPC-E clients that query a single server at the same time. We examine two cases, the first one being several Azure VMs running clients against Azure SQL Database -up to 64- and the second one includes several clients -up to 6- querying our local SQL Server 2008 across the LAN. We compare the average transaction time for each client thread running concurrently with several others with the single thread client performance. Our first result is Figure 9. The graph shows that the Azure Database does not take such a big performance hit as the local one. 6 concurrent local clients achieve a similar slowdown as 40 cloud clients. In this case, the cloud service performs better than our local SQL Server when we scale the number of concurrent clients.

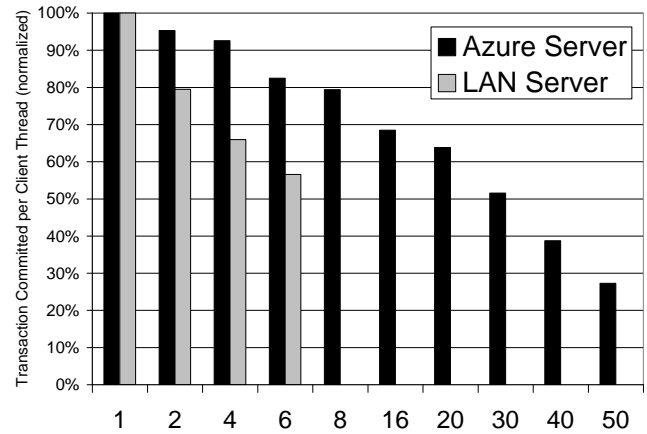


Figure 8: Transactions successfully executed as a function of the number of concurrent TPC-E clients.

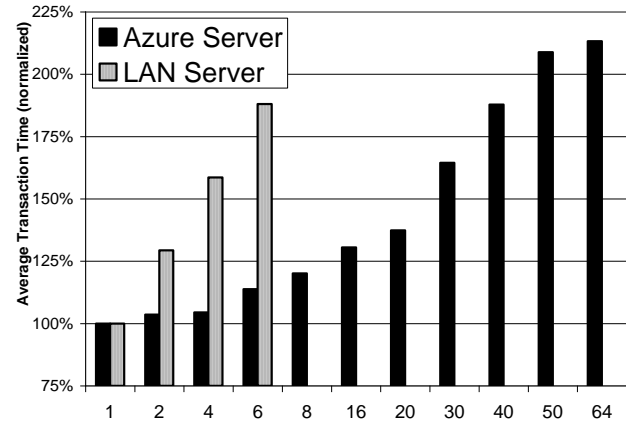


Figure 9: Average Transaction slowdown as a function of the number of concurrent clients.

We can also notice that the slowdown plateaus around 50 clients in the cloud. This does not indicate that the scalability is unlimited, since we are taking into account for this graph only the transactions that successfully commit. Thus, for a high number of threads trying to access the database concurrently, both SQL Azure Database and SQL Server 2008 start rejecting the incoming connections to the database. We present some data related to this phenomenon in Figure 8. Here we present, for each thread, the

normalized number of transactions that successfully commit. For example, each of the 64 concurrent cloud clients commits 17% of the transactions that 1 single client commits in the same time interval. Thus, the amount of work done is equivalent to 11 ideal concurrent clients $-64 * 17\% = 1088\%$. Again, the local server performs worse than the cloud server in this metric too, where 6 concurrent local clients show the same performance degradation as 20 to 30 concurrent cloud clients.

With these results in hand, one may argue that such performance degradation even at a relatively low number of clients -64 invalidates the scalability promises of the cloud. We should factor into account that every application that relies on a central server is going to hit a limit. Furthermore, for complex OLTP such as the TPC-E benchmark this limit is going to be lower, as we have seen. The system designer should take those limits into account, and design the application and databases accordingly. For example, we could have had partitioned our database into different databases, so several servers would share the load. Data partition is completely application dependent, though, so it falls outside the scope of our analysis.

5.3 Availability

One of the central promises of the SQL Azure Database is high availability. It is a common claim that the automatic management and replication of the database server in Azure is a superior alternative to a local in-house solution. We can automatically deploy new instances of databases in the cloud in a matter of seconds, which do not require setups, patches or update downtimes. Although these instances are available, there are no claims about the performance of the database server over time. Thus, we have scheduled a modified TPC-E client which runs locally and queries the SQL Azure Database for 20 minutes every hour. This client has been modified to be completely deterministic, because in such a short amount of time the probabilistic nature of the TPC-E benchmark distorts the results. Thus, we perform the exact same read only queries every hour. Figure 10 and Figure 11 summarize the results over several weeks: we started a TPCE client on our LAN -Figure 11- on November 25th, 2009 that run till January 23rd, 2010; we also started a TPCE client on a worker Role in Windows Azure - Figure 10- on December 18th, 2009 which run till January 31st, 2010.

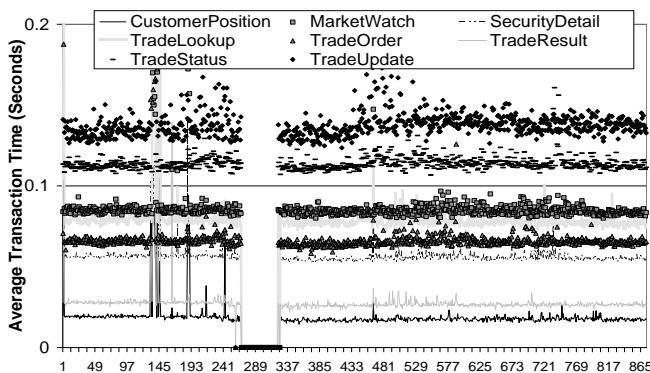


Figure 10: Performance of each TPC-E transaction over time in seconds, where the TPCE client runs in Azure.

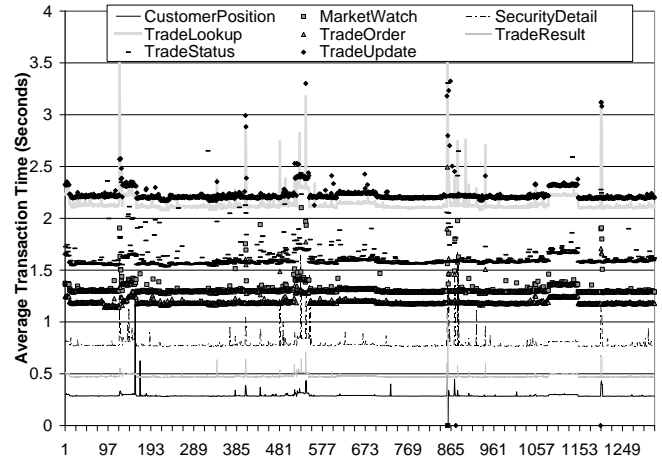


Figure 11: Performance of each TPC-E transaction over time in seconds, where the TPCE client runs in our LAN.

The results are, in general, stable. There are, however, some spikes in the graph that represent instances in which SQL Azure Database has been slower than usual to respond to our client, although these peaks are rather unusual. Also, our client in Windows Azure failed to contact the database server, and for several days the Azure fabric was unable to recycle it correctly. This period of time can be seen in Figure 10 between the 260 - December 29th - and 330 data points. Manual redeployment of the TPC-E client fixed this error, but the developers are advised to monitor continuously the health of each instance running in Azure, even if it is apparently running correctly -i.e., generating output blobs-.

6. RECOMMENDATIONS

In this section, we present our recommendation for developers and users of the Windows Azure cloud. These recommendations are based upon our experimental results and our experience developing scientific applications for Windows Azure.

6.1 Azure Storage Services

In order to improve performance users should choose a blob storage hierarchy so that accesses to the blobs are spread into as many storage partitions as possible. The data transfer throughput is sensitive to the number of concurrent accesses to a single partition. For example, with 32 clients concurrently accessing the blobs -no prefix used in the blob names- under the same container, the per-client data transfer rate can be degraded as much as 50% of that when only a single client is accessing the blobs.

The blob storage download bandwidth, when accessed from small instance types, is limited by the client's bandwidth for small numbers of concurrent clients -1 to 8 ; we saw a 100Mbit/s - $\sim 13\text{MB/s}$ - limitation. We have observed a per-client bandwidth drop of approx 1.5MB/s when we doubled the number of concurrent clients. The maximum service-side bandwidth achievable against a single blob for a high number of concurrent clients is limited to approximately 400MB/s -just about what we would expect from three 1Gb/s links if a blob is triple-replicated-. Therefore, we recommend using some extra data caching mechanisms on the worker role level to expand the per-client

bandwidth limit, and creating data replications on the blob storage to expand the server-side bandwidth limit.

In order to get the best performance out of the Table service, the table entities should be accessed by using partition keys and row keys only. Particularly, users should avoid querying tables using property filters under performance-critical or large concurrency circumstances. Currently, all tables are indexed on the PartitionKey and RowKey properties of each entity, but creating an index on any other properties cannot be specified. Under high concurrency circumstances, situations become even worse. In one of our experiments, over a half of the 32 concurrent clients got time-out exceptions instead of correct results when querying the same table partition -with ~220000 entities pre-populated- using property filters.

Table storage is not the same as a relational database and schemas/designs that work well in an RDBMS are typically not efficient in the Table Storage architecture. Replicating data between tables rather than doing a 'join' type operation in the client is often preferable as it minimizes calls to the Table Service.

Queues can be used for storage of many small objects as long as an approximate FIFO model is sufficient. We have not observed that queue performance for a single client is dependent on queue size -- we found consistent Add, Peek, and Receive operation performance from queues ranging in size from 200K messages to 2M messages.

Multiple queues should be used for supporting many concurrent readers/writers as we found that performance degraded as concurrent readers and/or writers were added, but each client obtained on average more than 10 operations per second for message sizes of 512 bytes to 8 KB for up to 32 writers. With 16 or fewer writers each client obtained 15-20 ops/sec. We also found that message retrieval was more affected by concurrency than message put operations so users cannot assume similar scale at each end of the queue.

6.2 Dynamic Scaling

Dynamically adding VMs to a deployment at runtime is a useful feature of Azure enabling dynamic load matching, but you should be aware that it often takes on the order of 10 minutes from the time of the request until the instance is integrated into the deployment. If fast scaling out is important, hot-standbys may be required if a 10 minute delay is not acceptable, although this option would incur a higher economic cost. Additionally, web roles took, on average, 60 seconds longer to come up for small instance types, and about 30 seconds longer for medium to extra large instance sizes.

6.3 Azure SQL Services

Users should expect a single Azure-based client accessing SQL Azure to take about twice as long as a single local-enterprise client accessing SQL Server within the enterprise -commodity hardware. Therefore, if speed is a chief concern and the number of concurrent clients is expected to be small, a local deployment -with local clients- will obtain the better performance. The opaque and potentially changing nature of the Cloud prevents us from determining exactly why there was a 2X slowdown.

We found that the throughput of a single large database instance - 10GB max size- peaks at 30 concurrent clients running

continuous transactions tested using OLTP-style TCPE database benchmark. The corresponding peak was seen at 6 concurrent clients in our LAN testing. In that case we experience 50% transaction failure and the average transaction completion time, for those that did complete, was 65% longer than that of a single client. For 8 concurrent clients we found a reasonable 15% transaction completion time increase. Use these numbers and the graphs presented in Section 5 as a general guide to concurrency limits in SQL Azure.

We did, however, find that performance over time was consistent, although there are rare occasions -less than 1.6% occurrence for 50% slowdown or worse, 0.3% for 2x slowdown or worse- where performance degraded significantly. Therefore, we have seen no reason to provision assuming a worst-case behavior in SQL Azure that is significantly worse than average-case behavior.

6.4 Testing & Development

Development on the Azure Development Fabric & Development Storage should be approached with realistic expectations. The Azure Development Fabric & Development Storage provides the ability to test connectivity and functional correctness of your own code, but only under limited concurrency. Additionally, the behavior of some SDK components may not match the deployment environment. One example is the load balancing algorithm: the Development Fabric appears to load-balance based on each instance's outstanding requests and does round-robin on equivalently loaded instances. In the full deployment environment the load balancer appears to utilize random scheduling amongst web role instances. Differences like these can make debugging deployed applications difficult if users assume the same behavior as seen in the smaller-scale local SDK environment.

7. CONCLUSION

In this paper we have presented the results from experiments we have conducted on Windows Azure. We have shown an exhaustive performance evaluation of each of the integral parts of the platform: virtual machines, storage services - table, blob and queue- and SQL services. Based on these experiments, we also provide our performance-related recommendations for users of the Windows Azure platform.

These cloud services are the building blocks for cloud applications, and are usually presented to the user as a black box, with no performance guarantees. Our main focus is to provide the community with performance information and concrete recommendations that help the design and development of scalable cloud applications.

8. REFERENCES

- [1] Amazon. Amazon Elastic Compute Cloud (EC2). <http://aws.amazon.com/ec2/>
- [2] Armbrust, M. and Fox, A. and Griffith, R. and Joseph, A.D. and Katz, R.H. and Konwinski, A. and Lee, G. and Patterson, D.A. and Rabkin, A. and Stoica, I. and others. Above the clouds: A Berkeley view of cloud computing. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28.
- [3] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. 2003. Xen and the art of virtualization. In Proceedings of the Nineteenth

- ACM Symposium on Operating Systems Principles (Bolton Landing, NY, USA, October 19 - 22, 2003). SOSP '03. ACM, New York, NY, 164-177. DOI=<http://doi.acm.org/10.1145/945445.945462>
- [4] Buyya, R., Chee Shin Yeo, Srikumar Venugopal, James Broberg, Ivona Brandic, Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility, *Future Generation Computer Systems*, Volume 25, Issue 6, June 2009, Pages 599-616, ISSN 0167-739X, DOI: 10.1016/j.future.2008.12.001.
 - [5] Deelman, E., Singh, G., Livny, M., Berriman, B., and Good, J. 2008. The cost of doing science on the cloud: the Montage example. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing* (Austin, Texas, November 15 - 21, 2008). *Conference on High Performance Networking and Computing*. IEEE Press, Piscataway, NJ, 1-12.
 - [6] Dejun, J. and Pierre, G. and Chi, C.H. 2009. EC2 Performance Analysis for Resource Provisioning of Service-Oriented Applications. In *The 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing* (Stockholm, Sweden, November 23, 2009)
 - [7] Evangelinos, C. and Hill, C. N. Cloud Computing for Parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. In *Proceedings of Cloud Computing and Its Applications* (Chicago, USA, October 22-23, 2008). CCA'08.
 - [8] Garfinkel, S. L. 2007. An evaluation of Amazon's grid computing services: EC2, S3 and SQS. Center for Research on Computation and Society School for Engineering and Applied Sciences, Harvard University, Tech. Rep., 2007
 - [9] Google. Google App Engine. <http://code.google.com/appengine/>
 - [10] Hill, Z. and Humphrey, M. 2009. A Quantitative Analysis of High Performance Computing with Amazon's EC2 Infrastructure. In *Proceedings of the 10th IEEE/ACM International Conference on GridComputing* (Banff, Canada, October 13-15 2009). Grid 2009.
 - [11] C. Hoffa, G. Mehta, T. Freeman, E. Deelman et al. 2008. On the Use of Cloud Computing for Scientific Workflows. *IEEE Fourth International Conference on eScience* (Indiana, USA, December 10 - 12, 2008). eScience 2008.
 - [12] Menon, A., Santos, J. R., Turner, Y., Janakiraman, G. and Zwaenepoel, W. 2005. Diagnosing performance overheads in the Xen virtual machine environment. In *Proceedings of the 1st ACM/USENIX international Conference on Virtual Execution Environments* (Chicago, IL, USA, June 11 - 12, 2005). VEE '05. ACM, New York, NY, 13-23. DOI=<http://doi.acm.org/10.1145/1064979.1064984>
 - [13] Microsoft. Windows Azure Platform. <http://www.microsoft.com/azure/default.mspx>
 - [14] Palankar, M. R., Iamnitchi, A., Ripeanu, M., and Garfinkel, S. 2008. Amazon S3 for science grids: a viable solution? In *Proceedings of the 2008 international Workshop on Data-Aware Distributed Computing* (Boston, MA, USA, June 24 - 24, 2008). DADC '08. ACM, New York, NY, 55-64. DOI=<http://doi.acm.org/10.1145/1383519.1383526>
 - [15] Ranadive, A., Kesavan, M., Gavrilovska, A., and Schwan, K. 2008. Performance implications of virtualizing multicore cluster machines. In *Proceedings of the 2nd Workshop on System-Level Virtualization For High Performance Computing* (Glasgow, Scotland, March 31 - 31, 2008). HPCVirt '08. ACM, New York, NY, 1-8. DOI=<http://doi.acm.org/10.1145/1435452.1435453>
 - [16] Walker, E. Benchmarking Amazon EC2 for High-Performance Scientific Computing. *Login: The Usenix Magazine*, Vol. 33, No. 5., 2008
 - [17] Youseff, L., Wolski, R., Gorda, B., and Krintz C. Evaluating the Performance Impact of Xen on MPI and Process Execution in HPC Systems. In *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing* (VTDC 2006)