Cloud Computing: Theory and Practice
Solutions to Exercises and Problems

Dan C. Marinescu

July 8, 2013

Contents

1 Chapter 1 - Preliminaries

Problem 1. *Mobile devices could benefit from cloud computing; explain the reasons you think that this statement is true or provide arguments supporting the contrary. Discuss several cloud applications for mobile devices; explain which one of the three cloud computing delivery models, SaaS, PaaS, or IaaS, would be used by each one of the applications and why.*

Mobile devices, such as smart phones and tablets, have limited power reserves thus, cannot carry out intensive computations or store excessive amounts of data; such devices can take advantage of high-speed WiFi connection to access data stored on a cloud and steer computations running on a cloud. Data sharing is a another major reason for mobile cloud applications; images, videos, music, or any other type of data stored on a cloud can be accessed by any device connected to the Internet and shared with others.

Videos and music can be streamed from a cloud supporting the SaaS delivery model. Massive Multi-player Online Games (MMOG) which require significant computing resources could be hosted by a cloud and accessed from tablets, smart phones or any other type of mobile devices. Scientific data processed and stored on a cloud can be shared by a community of researchers. Massive amounts of data can be processed on platforms supporting PaaS or IaaS cloud delivery models.

The latest WiFi technology, 802.11ac, with speeds of up to 1.3 Gbps, will most certainly lead to an increase of mobile applications of cloud computing. As the functionality and the diversity of sensors embedded in mobile devices increasers, we expect new mobile applications in areas as diverse as healthcare, education, entertainment, security, commerce, traffic monitoring, social networks, and so on.

Problem 2. *Do you believe that the homogeneity of a large-scale distributed systems is an advantage? Discuss the reasons for your answer. What aspects of hardware homogeneity are the most relevant in your view and why? What aspects of software homogeneity do you believe are the most relevant and why?*

Yes, homogeneity offers distinct advantages to the providers of computing resources and to the users of the systems.

It is expensive and technically challenging to maintain a large number of systems with different architecture, system software, and account management policies. Software and hardware updates are labor intensive. It is far from trivial to offer the user community a friendly environment and support fault-tolerance and other highly desirable system attributes. In such a heterogeneous environment data migration may require conversion and process migration could be extremely challenging. The users have to make sure that their applications use similar compilers and application libraries when running on different systems.

Hardware homogeneity reduces maintenance costs and may also reduce the initial investment cost as large numbers of systems are acquired from the same vendor. When all systems are running identical or similar system software, software maintenance can be automated and updates can be applied at the same time; this reduces the security exposure of the systems. The users benefit as the system offers a unique image. Data and process migration can be easily implemented both at the system and user level.

Problem 3. *Peer-to-peer systems and clouds share a few goals, but not the means to accomplish them. Compare the two classes of systems in terms of architecture, resource management, scope, and security.*

Peer-to-peer systems allow individuals to share data and computing resources, primarily storage space without sharing the cost of these resources; sharing other types of resources, such as computing cycles, is considerably more difficult. P2P systems have several desirable properties [67]:

- Require a minimally dedicated infrastructure, since resources are contributed by the participating systems.
- Are highly decentralized.
- Are scalable; the individual nodes are not required to be aware of the global state.
- Are resilient to faults and attacks, since few of their elements are critical for the delivery of service and the abundance of resources can support a high degree of replication.
- Individual nodes do not require excessive network bandwidth the way servers used in case of the client-server model do.
- The systems are shielded from censorship due to the dynamic and often unstructured system architecture.

Decentralization raises the question of whether P2P systems can be managed effectively and provide the security required by various applications. The fact that they are shielded from censorship makes them a fertile ground for illegal activities, including distribution of copyrighted content.

A peer-to-peer system is based on an ad-hoc infrastructure where individuals contribute resources to the system, but join and leave the system as they wish; thus, a high degree of redundancy is necessary, the information must be replicated on multiple sites. A peer-to-peer system maintains a directory of data and sites; this directory can be at a central site or distributed.

The need to make computing and storage more affordable and to liberate users from the concerns regarding system and software maintenance reinforced the idea of concentrating computing resources in large cloud computing centers; the users pay only for the resources they use thus, the name *utility computing*. In both cases the resources are accessed though the Internet and the applications are based on a client-server model with a thin client. The defining attributes of the cloud computing paradigm are:

- Cloud computing uses Internet technologies to offer elastic services. The term *elastic computing* refers to the ability to dynamically acquire computing resources and support a variable workload. A cloud service provider maintains a massive infrastructure to support elastic services.
- The resources used for these services can be metered and the users can be charged only for the resources they use.
- Maintenance and security are ensured by service providers.
- Economy of scale allows service providers to operate more efficiently due to specialization and centralization.

- Cloud computing is cost-effective due to resource multiplexing; lower costs for the service provider are passed on to the cloud users.
- The application data is stored closer to the site where it is used in a device- and location-independent manner; potentially, this data storage strategy increases reliability and security and, at the same time, it lowers communication costs.

A cloud computing center is managed by a cloud service provider which owns the servers and the networking infrastructure. Services are provided based on SLAs (Service Level Agreement). A cloud provides reliable storage and the data is automatically replicated. The level of user control of the cloud resources is different for the three cloud delivery models.

Security is a major concern in both cases, especially in the case of cloud computing.

Problem 4. *Compare the three cloud computing delivery models, SaaS, PaaS, and IaaS, from the point of view of the application developers and users. Discuss the security and the reliability of each one of them. Analyze the differences between the PaaS and the IaaS.*

Software-as-a-Service (SaaS) gives the capability to use applications supplied by the service provider in a cloud infrastructure. The applications are accessible from various client devices through a thin-client interface such as a Web browser (e.g., Web-based email). The user does not manage or control the underlying cloud infrastructure, including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

SaaS is not suitable for applications that require real-time response or those for which data is not allowed to be hosted externally. The most likely candidates for *SaaS* are applications for which:

- Many competitors use the same product, such as email.
- Periodically there is a significant peak in demand, such as billing and payroll.
- There is a need for Web or mobile access, such as mobile sales management software.
- There is only a short-term need, such as collaborative software for a project.

Platform-as-a-Service (PaaS) gives the capability to deploy consumer-created or acquired applications using programming languages and tools supported by the provider. The user does not manage or control the underlying cloud infrastructure, including network, servers, operating systems, or storage. The user has control over the deployed applications and, possibly, over the application hosting environment configurations. Such services include session management, device integration, sandboxes, instrumentation and testing, contents management, knowledge management, and Universal Description, Discovery, and Integration (UDDI), a platform-independent Extensible Markup Language (XML)-based registry providing a mechanism to register and locate Web service applications.

PaaS is not particulary useful when the application must be portable, when proprietary programming languages are used, or when the underlaying hardware and software must be customized to improve the performance of the application. The major *PaaS* application areas are in software development where multiple developers and users collaborate and the deployment and testing services should be automated.

Infrastructure-as-a-Service (IaaS) offers the capability to provision processing, storage, networks, and other fundamental computing resources; the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of some networking components, such as host firewalls. Services offered by this delivery model include: server hosting, Web servers, storage, computing hardware, operating systems, virtual instances, load balancing, Internet access, and bandwidth provisioning.

The *IaaS* cloud computing delivery model has a number of characteristics, such as the fact that the resources are distributed and support dynamic scaling, it is based on a utility pricing model and variable cost, and the hardware is shared among multiple users. This cloud computing model is particulary useful when the demand is volatile and a new business needs computing resources and does not want to invest in a computing infrastructure or when an organization is expanding rapidly.

Problem 5. *Overprovisioning is the reliance on extra capacity to satisfy the needs of a large community of users when the average-to-peak resource demand ratio is very high. Give an example of a large-scale system using overprovisioning and discuss if overprovisioning is sustainable in that case and what are the limitations of it. Is cloud elasticity based on overprovisioning sustainable? Give the arguments to support your answer.*

The Internet is probably the best known example of a large-scale system based on overprovisioning. Indeed, the backbones of different ISPs (Internet Service Providers) are high-capacity fiber optic channels able to support a very high volume of network traffic, orders of magnitude larger than the average traffic. This is economically feasible because the cost of a high-capacity fiber channel is only a fraction of the total cost for installing the fiber. Nevertheless, the Internet bandwidth available to individual customers is limited by the so called "last mile" connection, the last link connecting a home to a node of ISP's network. Another important aspect is that the Internet is based on a "Best Effort" paradigm thus, the routers of a heavily loaded network are allowed to drop packets.

Overprovisioning in a computer cloud is a consequence of the commitment to support an "elastic" computing environment, in other words, to supply additional resource to the users when they are needed. Thus, the CSPs (Cloud Service Providers) maintain an infrastructure considerably larger than the one capable to respond to the average user needs, and the average server utilization is low, say in the range to 20-40 %. The cost for supporting such an infrastructure cannot be economically justified. This cost has two basic components, the initial investment and the recurring costs for energy consumption, maintenance, and so on. Arguably, the cost for the initial investment cannot be reduced if the CSP is really committed to cloud elasticity, but can be controlled if the SLAs require a user to specify both the average and the maximum level of resource consumption. Then the cloud resource management should implement an admission control policy to ensure that the system can satisfy all its existing commitments and does not have to pay penalties for breaking the contractual agreements. The recurring costs can be reduced by migrating applications from lightly loaded systems and by turning them to a sleep mode to reduce power consumption.

Problem 6. *Discuss the possible solution for stabilizing cloud services mentioned in [26] inspired by BGP (Border Gateway Protocol) routing [37, 84].*

In the paper "Icebergs in the Clouds: the Other Risks of Cloud Computing" B. Ford expresses a concern related to instability risks from interacting services: "Cloud services and applications increasingly build atop one another in ever more complex ways, such as cloud based advertising or mapping services used as components in other, higher-level cloud-based applications, all of these building on computation and storage infrastructure offered by still other providers. Each of these interacting, code-pendent services and infrastructure components is often implemented, deployed, and maintained independently by a single company that, for reasons of competition, shares as few details as possible about the internal operation of its services."

The author suggests: " One place we might begin to study stability issues between interacting cloud services, and potential solutions, is the extensive body of work on the unexpected inter-AS (Autonomous System) interactions frequently observed in BGP routing[37, 84]. In particular, the "dependency wheel" model, useful for reasoning about BGP policy loops, seems likely to generalize to higher-level control loops in the cloud, such as load balancing policies. Most of the potential solutions explored so far in the BGP space, however, appear largely specific to BGPor at least to routingand may have to be rethought "from scratch" in the context of more general, higher-level cloud services. Beyond BGP, classic control theory may offer a broader source of inspiration for methods of understanding and ensuring cloud stability. Most conventional control-theoretic techniques, however, are unfortunately constructed from the assumption that some "master system architect" can control or at least describe all the potentially-interacting control loops in a system to be engineered. The cloud computing model violates this assumption at the outset by juxtaposing many interdependent, reactive control mechanisms that are by nature independently developed, and are often the proprietary and closely-guarded business secrets of each provider."

The ever increasing depth of the software stack is likely to lead to new security and performance concerns. At the same time, a realistic standardization effort runs contrary to the interests of cloud service providers.

Problem 7. *An organization debating whether to install a private cloud or to use a public cloud, e.g., the AWS, for its computational and storage needs, asks your advice. What information will you require to base your recommendation on, and how will you use each one of the following items: (a) the description of the algorithms and the type of the applications the organization will run; (b) the system software used by these applications; (c) the resources needed by each application; (d) the size of the user population; (e) the relative experience of the user population; (d) the costs involved.*

Public clouds have distinct cost advantages over private clouds; there is no initial investment in the infrastructure, no recurring costs for administration, maintenance, energy consumption, and for the user support personnel. The main concern is security and privacy. An organization with very strict security and privacy concerns is very unlikely to use a public cloud.

The type of applications play a critical role, scientific and engineering computations which require a low latency interconnection network and enjoy only fine-grain parallelism are unlikely to fare well on either a public or a private cloud. A large user population is more likely to use identical or similar software and to cooperate by sharing the raw data and the results; thus, a private cloud seems more advantageous in this case. Some of the services offered by private clouds target experiences users, e.g., AWS services such as *ElasticBeanstalk*, while others are accessible to lay persons.

Problem 8. *A university is debating the question in Problem 7. What will be your advice and why? Should software licensing be an important element of the decision?*

Both education and research are university activities that could benefit from cloud computing. A public cloud is probably an ideal environment for education; indeed, security is not a significant concern for computing related to education. The student computational activity peaks when projects and other assignments are due and it is relatively low during the remaining of a semester thus, the investment into a large infrastructure needed for brief periods of time do not seem to be justified. Moreover, once a student learns how to use a cloud service, e.g., AWS, this skill could help in finding a job and then during his/her employment. The students share the software for their assignments and a public cloud could provide an ideal environment for group projects. Software licensing can be a problem as some projects may require the use of licensed software.

The benefits of a private cloud for supporting university research is increasingly more questionable as the cost of the hardware decreases and the computational needs of many research groups can be met with a medium or a high-end server which costs only a few thousand dollars. Of course, some research groups may need access to supercomputers but acquiring a supercomputer is only justified for a very few well-funded research universities.

Problem 9. *An IT company decides to provide free access to a public cloud dedicated to higher education. Which one of the three cloud computing delivery models, SaaS, PaaS, or IaaS should it embrace and why? What applications would be most beneficial for the students? Will this solution have an impact on distance learning? Why?*

A public cloud dedicated to higher education would be extremely beneficial; it would allow concentration of educational resources and sharing of software for different educational activities. It would increase the student participation and interest in the different topics. At the same time, it would significantly reduce the cost of educational computing.

Each one of the three cloud delivery models would have its own applications. For example, *SaaS* could be uses for testing and for systems like *Piazza* (see https://piazza.com/) which allow groups of students and faculty to interact. Collaborative environments which allow an instructor to interact with a large class where the students are connected via mobile devices would most certainly increase the level of student participation. Projects given to engineering and science students would have useful results as the students would be able to run concurrently multiple models of the systems and use different parameters of the models. Such projects would require either a *Paas* or an *IaaS* cloud model.

2 Chapter 2 - Basic Concepts

Problem 1. *Non-linear algorithms do not obey the rules of scaled speed-up. For example, it was shown that when the concurrency of an $\mathcal{O}(N^3)$ algorithm doubles, the problem size increases only by slightly more than 25%. Read [75] and explain this result.*

The author of [75] writes:"As a practical matter the scientific and commercial problems that are most in need of speedup are the so-called compute-bound problems since the I/O-bound problems would yield to better data transmission technology not more processing capability. These compute-bound problems are generally super-linear, typically exhibiting sequential time complexity in the range $\mathcal{O}(n^2)$ to $\mathcal{O}(n^4)$ for problems of size n. The reason $\mathcal{O}(n^4)$ problems are common in science and engineering is that they often model physical systems with three spatial dimensions developing in time. A frequent challenge with these problems is not to solve a fixed-size instance of the problem faster, but rather to solve larger instances within a fixed time budget. The rationale is simple: The solutions are so computationally intensive that problem instances of an interesting size do not complete execution in a tolerable amount of time. Here "tolerable" means the maximum time the machine can be monopolized or the scientist can wait between experiments. Thus, with present performance the scientist solves a smaller-than-desirable problem to assure tolerable turnaround. The fact that the tolerable turnaround time will remain constant implies that increased performance will be used to attack larger problems.

What is the effect of parallelism on problem size? Keeping time fixed at t and (unrealistically) assuming the best possible speedup, it follows that super-linear problems can be improved only sub-linearly by parallel computation. Specifically, if a problem takes

$$t = cn^x \tag{1}$$

sequential steps, and if the application of p processors permits an increase in the problem size by a factor of m

$$t = c(mn)^x/p \tag{2}$$

the problem size can increase by at most a factor of

$$m = p^{1/x} \tag{3}$$

For example, to increase by two orders of magnitude the size of a problem whose sequential performance is given by

$$t = cn^4 \tag{4}$$

requires, optimistically, $100,000,000$ processors."

Problem 2. *Given a system of four concurrent threads t_1, t_2, t_3 and t_4 we take a snapshot of the consistent state of the system after $3, 2, 4$, and 3 events in each thread, respectively; all but the second event in each thread are local events. The only communication event in thread t_1 is to send a message to t_4 and the only communication event in thread t_3 is to send a message to t_2. Draw a space-time diagram showing the consistent cut; mark individual events on thread t_i as e_i^j.*

How many messages are exchanged to obtain the snapshot in this case? The snapshot protocol allows the application developers to create a checkpoint. An examination of the checkpoint data shows that an error has occurred and it is decided to trace the execution. How many potential execution paths must be examined to debug the system?

Figure 1 shows the space-time diagram and the consistent cut $C_1 = (e_1^3, e_2^2, e_3^4, e_4^3)$.

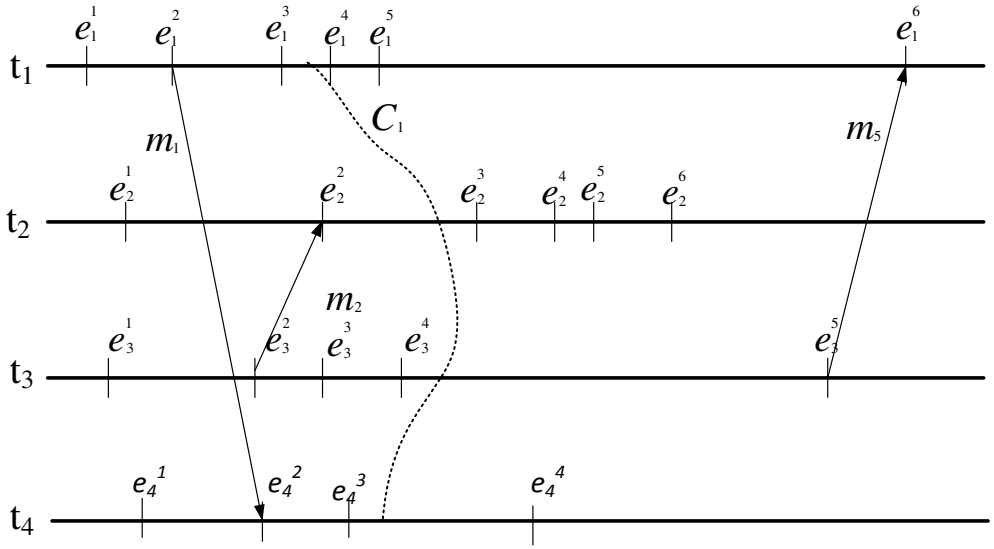

Figure 1: Space-time diagram for four concurrent threads t_1, t_2, t_3 and t_4 when all, but the second event in each thread, are local. The only communication event in thread t_1 is to send a message to t_4 and the only communication event in thread t_3 is to send a message to t_2. The cut $C_1 = (e_1^3, e_2^2, e_3^4, e_4^3)$ is consistent, it does not include non-causal events, e.g., the receiving of a message without the event related to the sending of the message.

The snapshot protocol of Chandy and Lamport consists of three steps [16]

1. Process p_0 sends to itself a "take snapshot" message.

2. Let p_f be the process from which p_i receives the"take snapshot" message for the first time. Upon receiving the message, the process p_i records its local state, σ_i, and relays the "take snapshot" along all its outgoing channels without executing any events on behalf of its underlying computation; channel state $\xi_{f,i}$ is set to empty and process p_i starts recording messages received over each of its incoming channels.

3. Let p_s be the process from which p_i receives the "take snapshot" message beyond the first time; process p_i stops recording messages along the incoming channel from p_s and declares channel state $\xi_{s,i}$ as those messages that have been recorded.

Each"take snapshot" message crosses each channel exactly once and every process p_i has made its contribution to the global state; a process records its state the first time it receives a "take snapshot" message and then stops executing the underlying computation for some

time. Thus, in a fully connected network with n processes the protocol requires $n \times (n-1)$ messages, one on each channel.

In our case we have $n = 4$ threads, t_1, t_2, t_3 and t_4 , thus the number of messages is

$$M_{msg} = n(n-1) = 4 \times 3 = 12. \tag{5}$$

The number of events to reach the state $(3, 2, 4, 3)$ is

$$n = \frac{(3+2+4+3)!}{(3!+2!+4!+3!)} = \frac{12!}{38}. \tag{6}$$

Problem 3. *The run-time of a data-intensive application could be days, or possibly weeks, even on a powerful supercomputer. Checkpoints are taken for a long-running computation periodically and when a crash occurs the computation is restarted from the latest checkpoint. This strategy is also useful for program and model debugging; when one observes wrong partial results the computation can be restarted from a checkpoint where the partial results seem to be right.*

Express η, the slowdown due to checkpointing, for a computation when checkpoints are taken after a run lasting τ units of time and each checkpoint requires κ units of time. Discuss optimal choices for τ and κ.

The checkpoint data can be stored locally, on the secondary storage of each processor, or on a dedicated storage server accessible via a high-speed network. Which solution is optimal and why?

Assume that the computation requires n runs, each of duration τ units of time, thus the running time without checkpointing is

$$T_{nc} = n \times \tau. \tag{7}$$

Then the running time with checkpointing is

$$T_c = b \times (\tau + \kappa). \tag{8}$$

The slowdown is the ratio of the running time with checkpointing over the running time without

$$\eta = \frac{T_c}{T_{nc}} = \frac{\tau + \kappa}{\tau} = 1 + \frac{\kappa}{\tau}. \tag{9}$$

The slowdown is a function of the ratio $\frac{\kappa}{\tau}$. The time required for checkpointing, κ, is a function of the size of the snapshot thus, it is fixed for a given application. The only means to affect the slowdown is the choice of τ.

For a given κ, the smaller the ratio, the lower the slowdown, but the larger the time between two consecutive snapshots thus, the more significant the loss in case of a system failure. Assuming that a system failure in the interval τ is uniformly distributed, then the average time lost in case of a failure is $\tau/2$. If $\tau = \kappa$ then $\eta = 2$, the execution time doubles; if $\tau = 2\kappa$ then $\eta = 1.5$ the execution time increases by 50% and so on.

Checkpointing to a persistent storage device, e.g., disk, is I/O intensive, thus time consuming. In recent years, as the cost of the primary storage has decreased substantially, some advocate checkpointing to primary storage, a much faster alternative. But the primary storage

is volatile so the checkpoint is lost in case of a power failure. Also recovering the checkpoint data in case of a system failure could be challenging. An alternative is to checkpointing to the primary storage and then transferring the checkpoint data to persistent storage while the application is running.

The use of flash memories as the persistent storage will substantially reduce the time for writing the snapshots to persistent storage.

Problem 4. *What is in your opinion the critical step in the development of a systematic approach to all-or-nothing atomicity? What does a systematic approach means? What are the advantages of a systematic versus an ad-hoc approach to atomicity?*

The support for atomicity affects the complexity of a system. Explain how the support for atomicity requires new functions/mechanisms and how these new functions increase the system complexity. At the same time, atomicity could simplify the description of a system; discuss how it accomplishes this.

The support for atomicity is critical for system features which lead to increased performance and functionality such as: virtual memory, processor virtualization, system calls, and user-provided exception handlers. Analyze how atomicity is used in each one of these cases.

It is desirable to have a systematic approach rather than an ad-hoc one because atomicity is required in many contexts. It would be difficult to implement, difficult to use, and prone to error to have ad-hoc implementations of atomicity for each context it is used for. A systematic approach to atomicity must address several delicate questions:

1. How to guarantee that only one atomic action has access to a shared resource at any given time.

2. How to return to the original state of the system when an atomic action fails to complete.

3. How to ensure that the order of several atomic actions leads to consistent results.

There is not a single critical aspect in the development of a systematic approach to all-or-nothing atomicity; of course all atomic actions should not expose the state of the system until the action is completed and should be able to redo all actions carried out before the commit point and, at the same time, provide answers to the delicate questions mentioned above.

The first item requires some mechanism to serialize the access to shared data structures and other resources. This is commonly done by using locks, semaphores, or monitors. But each one of these choices has its own problems and solving these problems adds to system complexity. For example, locks could lead to deadlocks and the system must include a lock manager that attempts to avoid such undesirable situations. With locks and priority scheduling one can also experience priority inversion, another major headache. In recent years a substantial effort has been dedicated to lock-free data structures but this does not eliminate the need for locks or similar mechanisms.

The second item requires the system to implement a journal storage which would allow it tom roll back an action. The third item requires the implementation of *before-or-after atomicity*. This means that, from the point of view of an external observer, the effect of multiple actions is as if these actions have occurred one after another, in some order. This is non trivial and is case dependent. For example, a transaction acting on two accounts should either debit the first account and then credit the second one, or leave both accounts unchanged. The order is important, as the first account cannot be left with a negative balance.

Indeed, atomicity is critical for system features which lead to increased performance and functionality, such as virtual memory, processor virtualization, system calls, and user-provided exception handlers. The implementation of all system functions requires kernel data structures which should be protected from concurrent access which would leave the system in an inconsistent state. For example, a page fault could occur during the decoding of an instruction in a pipelined system. The actions of the scheduler, multi-level memory manager, and virtual memory must be well coordinated.

Problem 5. *The Petri Net in Figure 2(d), models a group of n concurrent processes in a shared-memory environment. At any given time only one process may write, but any subset of the n processes may read at the same time, provided that no process writes. Identify the firing sequences, the markings of the net, the postsets of all transitions and the presets of all places. Can you construct a state machine to model the same process?*

Petri Nets are able to model *exclusion*; for example, the net in Figure 2(d), models a group of n concurrent processes in a shared-memory environment. At any given time only one process may write, but any subset of the n processes may read at the same time, provided that no process writes. Place p_3 models the process allowed to write, p_4 the ones allowed to read, p_2 the ones ready to access the shared memory, and p_1 the running tasks. Transition t_2 models the initialization/selection of the process allowed to write and models t_1 of the processes allowed to read, whereas t_3 models the completion of a write and t_4 models the completion of a read. Indeed, p_3 may have at most one token while p_4 may have at most n. If all n processes are ready to access the shared memory, all n tokens in p_2 are consumed when transition t_1 fires. However, place p_4 may contain n tokens obtained by successive firings of transition t_2.

The markings of the net describe the disposition of tokens in places p_1, p_2, p_3, p_4, respectively, and can be used to describe the state of the system. For example, the initial marking of the net in Figure 2(d) is

$$M_0 = (n, n, 0, 0) \tag{10}$$

and in this state both `read` and `write` operations are permitted.

When transition t_2 fires - this represents the initialization of a `write` request - then the marking becomes

$$M_1 = (n-1, 0, 1, 0) \tag{11}$$

and neither transition t_1 - which represents the initialization of a `read` request - nor t_2 can fire, as place p_2 contains no token. This means that only one process can `write` at a time and no process can `read` during the `write` operation.

After transition t_3 fires - this represents the actual `write` operation - the marking becomes the initial one, M_0.

When the system is in state represented by marking M_0 and transition t_1 fires - this represents the initialization of a `read` request - then the marking becomes

$$M_1 = (n-1, n-1, 0, 1) \tag{12}$$

and transition t_2 cannot fire - no `write` can be initialized, but successive $n-1$ reads can be initiated and then completed. From the state represented by marking M_1 there are two possibilities:

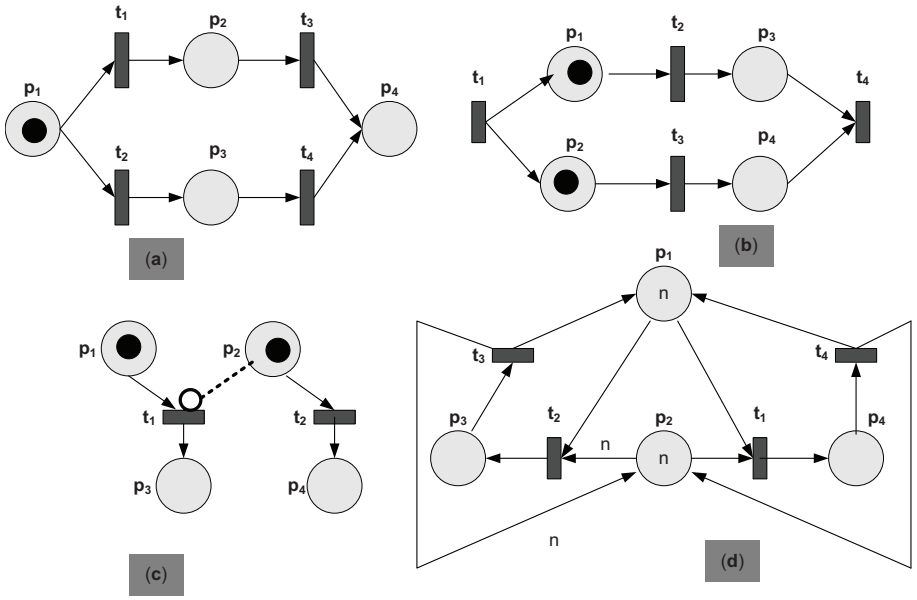

Figure 2: (a) A state machine; there is the choice of firing t_1, or t_2; only one transition fires at any given time, concurrency is not possible. (b) A *marked graph* can model concurrency but not choice; transitions t_2 and t_3 are concurrent, there is no causal relationship between them. (c) An extended net used to model priorities; the arc from p_2 to t_1 is an inhibitor arc. The process modeled by transition t_1 is activated only after the process modeled by transition t_2 is activated. (d) Modeling exclusion; transitions t_1 and t_2 model writing and, respectively, reading with n processes to a shared memory. At any given time only one process may write, but any subset of the n processes may read at the same time, provided that no process writes.

- t_4 fires and then the `read` operation is completed and the next state is represented by marking M_0.
- t_4 fires again and the marking becomes

$$M_1 = (n-2, n-2, 0, 2) \tag{13}$$

This process can be repeated $k < n$ times leading to the markings

$$M_{k-1} = (n-k, n-k, 0, k) \tag{14}$$

Several possible firing sequences:

- $(t_2t_3)^k \implies$ k consecutive `write` operations.
- $(t_1t_4)^k \implies$ k consecutive `read` operations.
- $((t_2t_3), (t_1t_4)^k) \implies$ a `write` operation followed by k consecutive `read` operations.
- $((t_2t_3)^n, (t_1t_4)^k, (t_2t_3)) \implies$ n consecutive `write` operation followed by k consecutive `read` operations, followed by a `write` operation.

The postsets of the transitions are:

- $t_1 \implies (P_4)$
- $t_2 \implies (P_3)$
- $t_3 \implies (P_1, P_2)$
- $t_4 \implies (P_1, P_2)$

The presets of the places are:

- $p_1 \implies (t_3, t_4)$
- $p_2 \implies (t_3, t_4)$
- $p_3 \implies (t_2)$
- $p_4 \implies (t_3)$

It is not possible to construct a state-machine model of the system because a state-machine does not support concurrency and we have to model the concurrent `read` operations.

Problem 6. *Explain briefly how the publish-subscribe paradigm works and discuss its application to services such as bulletin boards, mailing lists, and so on. Outline the design of an event service based on this paradigm, see Figure 3(b). Can you identify a cloud service that emulates an event service?*

Two sets of entities are involved in the publish-subscribe system, those who provide the information and those who consume it. The two sets can be disjoint, as is the case with a news service, or they can consist of the same entities, as is the case of bulletin boards and mailing lists. The actual operation of different publish-subscribe systems can be very different. For example in case of Facebook, an individual with a Facebook page chooses a set of individuals who have access to this information, the so called "friends," and publishes items of interest on his/her page and decides who has access to this information, everybody with a Facebook account or only the friends who are made aware whenever a new item is added to the page. In case of a mailing list individuals subscribe to the list, there is a moderator who decides what items can be distributed to all those who have subscribed to the list.

In case of an event service, the designer of the system defines a number of classes of events and then chooses the events to be included in each class. Once the event ontology is made public individuals subscribe to the events they are interested in. There is a queue of events in each class and once a new event is added to the queue for the particular event, a message is sent to notify all the subscribers for that type of event.

The SNS (Simple Notification Service) of AWS is an example of an *IaaS* cloud service based on the publish-subscribe paradigm. According to *http://aws.amazon.com/sns/* the steps required for using SNS are:

- Create a topic. A topic is an "access point" identifying a specific subject or event type for publishing messages and allowing clients to subscribe for notifications.
- Set policies for the topic. Once a topic is created, the topic owner can set policies for it, such as limiting who can publish messages or subscribe to notifications, or specifying which notification protocols will be supported (i.e. SQS, HTTP/HTTPS, email, SMS). A single topic can support notification deliveries over multiple transport protocols.

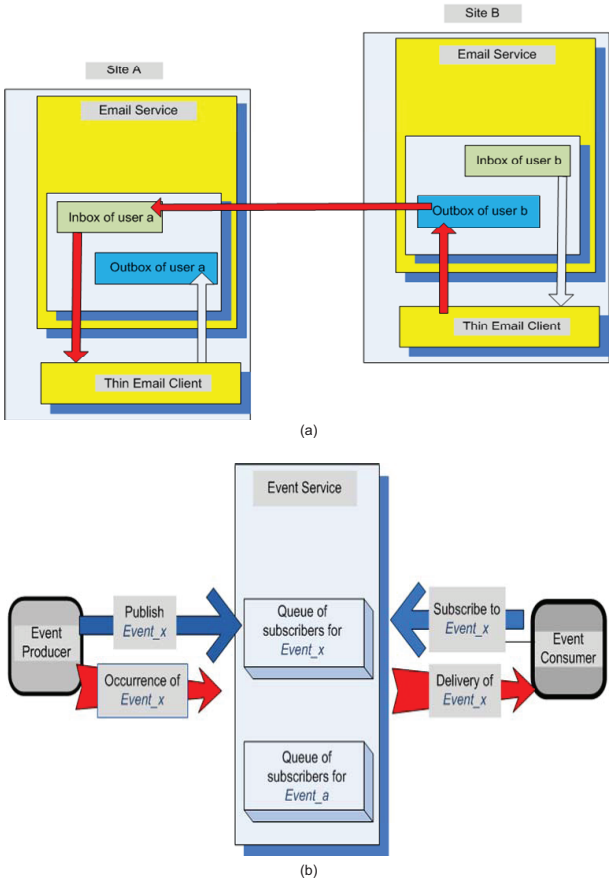

Figure 3: (a) Email service; the sender and the receiver communicate asynchronously using inboxes and outboxes. Mail demons run at each site. (b) An event service supports coordination in a distributed system environment. The service is based on the publish-subscribe paradigm; an event producer publishes events and an event consumer subscribes to events. The server maintains queues for each event and delivers notifications to clients when an event occurs.

- Add subscribers to a topic. Subscribers are clients interested in receiving notifications from topics of interest; they can directly subscribe to a topic or be subscribed by the topic owner. Subscribers specify the protocol format and end-point (SQS ARN, URL, email address, phone number, etc.) for notifications to be delivered. Upon receiving a subscription request, Amazon SNS will send a confirmation message to the specified end-point, asking the subscriber to explicitly opt-in to receiving notifications from that topic. Opting-in can be done by calling an API, using a command line tool, or for email notifications simply clicking on a link.

Problem 7. *Tuple spaces can be thought of as an implementation of a distributed shared-memory. Tuple-spaces have been developed for many programming languages including Java, Lisp, Python, Prolog, Smalltalk, and Tcl. Explain briefly how tuple spaces work. How secure and scalable are the tuple spaces you are familiar with, e.g., JavaSpaces?*

There are two groups of entities involved in a tuple space: producers which post the data in the tuple space and consumers which retrieve the data from the tuple spaces. Original

implementation of tuple space such as Linda[1] were neither secure nor scalable. Indeed, a tuple space is a place where a large number of users compete for access.

The site *http://river.apache.org/doc/specs/html/js-spec.html* describes the JavaSpaces application model as follows: "A *JavaSpaces* service holds entries. An entry is a typed group of objects, expressed in a class for the Java platform that implements the interface *net.jini.core.entry.Entry*. Entries are described in detail in the Jini Entry Specification. An entry can be written into a JavaSpaces service, which creates a copy of that entry in the space that can be used in future lookup operations. You can look up entries in a JavaSpaces service using templates, which are entry objects that have some or all of its fields set to specified values that must be matched exactly. Remaining fields are left as wildcards, these fields are not used in the lookup.

There are two kinds of lookup operations: `read` and `take`. A `read` request to a space returns either an entry that matches the template on which the read is done, or an indication that no match was found. A `take` request operates like a `read`, but if a match is found the matching entry is removed from the space. You can request a JavaSpaces service to notify you when an entry that matches a specified template is written. This is done using the distributed event model contained in the package *net.jini.core.event* and described in the Jini Distributed Events Specification.

All operations that modify a JavaSpaces service are performed in a transactionally secure manner with respect to that space. That is, if a write operation returns successfully, that entry was written into the space (although an intervening take may remove it from the space before a subsequent lookup of yours). And if a take operation returns an entry, that entry has been removed from the space, and no future operation will read or take the same entry. In other words, each entry in the space can be taken at most once. Note, however, that two or more entries in a space may have exactly the same value.

The architecture of JavaSpaces technology supports a simple transaction mechanism that allows multi-operation and/or multi-space updates to complete atomically. This is done using the two-phase commit model under the default transaction semantics, as defined in the package net.jini.core.transaction and described in the Jini Transaction Specification. Entries written into a JavaSpaces service are governed by a lease, as defined in the package *net.jini.core.lease* and described in the Jini Distributed Leasing Specification."

Problem 8. *Consider a computation consisting of n stages with a barrier synchronization among the N threads at the end of each stage. Assuming that you know the distribution of the random execution time of each thread for each stage show how one could use order statistics [21] to estimate the completion time of the computation.*

Given N random variables $X_1, X_2, \ldots\ldots, X_N$, the order statistics $X_{(1)}, X_{(2)}, ..., X_{(N)}$ are also random variables, defined by sorting the values (realizations) of $X_1, X_2, \ldots\ldots, X_N$ in increasing order. When the random variables $X_1, X_2, \ldots\ldots, X_N$ form a sample they are independent and identically distributed.

The first order statistic (or smallest order statistic) is the minimum of the sample

$$X_{(1)} = \min\{X_1, X_2, \ldots\ldots, X_N\} \tag{15}$$

Similarly, for a sample of size N, the N-th order statistic (or largest order statistic) is the maximum

[1]Linda language was developed by David Gelernter and Nicholas Carriero at Yale University

$$X_{(N)} = \max\{X_1, X_2, \ldots\ldots, X_N\} \tag{16}$$

If the random variables $X_1^k, X_2^k, \ldots\ldots, X_N^k$ represent the time required by threads $t_1, t_2, \ldots\ldots t_N$, respectively, to complete the execution in stage k of a computation $\mathcal{C}$ with M stages, then the completion time of stage k is given by the N-order statistics:

$$X_{(N)}^k = \max\{X_1^k, X_2^k, \ldots\ldots, X_N^k\} \tag{17}$$

as a barrier synchronization requires the threads to wait until the one which takes the longest time completes its execution.

Then the total time to completion of computation $\mathcal{C}$ with M stages is:

$$T_{\mathcal{C}} = \sum_{k=1}^{M} X_{(N)}^k. \tag{18}$$

Obviously, the order statistics $X_{(N)}$ depends on the distribution of the random variables. For example, for a (μ, σ) normal distribution the expected value of $X_{(N)}$ is bounded

$$E\left[X_{(N)}\right] \leq \mu + \frac{(N-1)\sigma}{(2N-1)^{1/2}}. \tag{19}$$

If the random variables $X_1^k, X_2^k, \ldots\ldots, X_N^k$ are normally distributed with mean μ_k and variance σ_k then in our example

$$T_{\mathcal{C}} \leq \sum_{k=1}^{M} \mu_k + \frac{(N-1)}{(2N-1)^{1/2}} \sum_{k=1}^{M} \sigma_k. \tag{20}$$

Problem 9. *In Section 4.9 we analyze cloud computing benchmarks and compare them with the results of the same benchmarks performed on a supercomputer. This is not unexpected; discuss the reasons why we should expect the poor performance of fine-grain parallel computations on a cloud.*

A recent paper [42] describes the set of applications used at NERSC (National Energy Research Scientific Computing Center) and presents the results of a comparative benchmark of *EC2* and three supercomputers.

The systems used for the comparison with cloud computing are:

Carver - a 400 node IBM iDataPlex cluster with quad-core Intel Nehalem processors running at 2.67 GHz and with 24 GB of RAM (3 GB/core). Each node has two sockets; a single Quad Data Rate (QDR) IB link connects each node to a network that is locally a fat-tree with a global two-dimensional mesh. The codes were compiled with the Portland Group suite version 10.0 of and Open MPI version 1.4.1.

Franklin - a 9 660-node Cray XT4; each node has a single quad-core 2.3 GHz AMD Opteron "Budapest" processor with 8 GB of RAM (2 GB/core). Each processor is connected through a 6.4 GB/s bidirectional HyperTransport interface to the interconnect via a Cray SeaStar-2 ASIC. The SeaStar routing chips are interconnected in a tri-dimensional torus topology, where each node has a direct link to its six nearest neighbors. Codes were compiled with the Pathscale or the Portland Group version 9.0.4.

Lawrencium - a 198-node (1 584 core) Linux cluster; a compute node is a Dell Poweredge 1950 server with two Intel Xeon quad-core 64 bit, 2.66 GHz Harpertown processors with 16 GB of RAM (2 GB/core). A compute node is connected to a Dual Data Rate InfiniBand network configured as a fat tree with a 3 : 1 blocking factor. Codes were compiled using Intel 10.0.018 and Open MPI 1.3.3.

The virtual cluster at Amazon had four *EC2* CUs (Compute Units), two virtual cores with two CUs (Compute Units) each, and 7.5 GB of memory (an `m1.large` instance in Amazon parlance); a Compute Unit is approximately equivalent to a $1.0 - 1.2$ GHz 2007 Opteron or 2007 Xeon processor. The nodes are connected with gigabit Ethernet. The binaries were compiled on Lawrencium. The results reported in [42] are summarized in Table 1.

Table 1: The results of the measurements reported in [42]

System	DGEMM Gflops	STREAM GB/s	Latency μs	Bndw GB/S	HPL Tflops	FFTE Gflops	PTRANS GB/s	RandAcc GUP/s
Carver	10.2	4.4	2.1	3.4	0.56	21.99	9.35	0.044
Frankl	8.4	2.3	7.8	1.6	0.47	14.24	2.63	0.061
Lawren	9.6	0.7	4.1	1.2	0.46	9.12	1.34	0.013
EC2	4.6	1.7	145	0.06	0.07	1.09	0.29	0.004

The results in Table 1 show us that

- The computing power measured in Tflops or Gflops of the *EC2* virtual cluster considerably lower than the one of the three supercomputers.
- The memory bandwidth of the *EC2* virtual cluster is almost an order of magnitude smaller than the one of the three supercomputers.
- The communication latency of the *EC2* virtual cluster is almost two order of magnitude larger than the one of the three supercomputers.
- The communication bandwidth of the *EC2* virtual cluster is almost two order of magnitude smaller than the one of the three supercomputers.

Communication intensive applications are affected by the increased latency (more than 70 times larger then *Carver*) and lower bandwidth (more than 70 times smaller than *Carver*). Computationally intensive applications are affected by lower CPU and memory bandwidth.

3 Chapter 3 - Infrastructure

Problem 1. *Several desirable properties of a large-scale distributed system includes transparency of access, location, concurrency, replication, failure, migration, performance, and scaling. Analyze how each one of these properties applies to AWS.*

We discuss these properties in the context of several AWS services. *EC2* is a Web service with a simple interface for launching instances of an application under several operating systems. A user can:

- launch an instance from an existing AMI and terminate an instance;
- start and stop an instance;
- create a new image;
- add tags to identify an image; and
- reboot an instance.

EC2 allows the import of virtual machine images from the user environment to an instance through a facility called *VM import.* It also automatically distributes the incoming application traffic among multiple instances using the *elastic load-balancing* facility. The service ensures transparency of access, location, concurrency, replication, failure, migration, performance, and scaling. Indeed, the user is not concerned with the location of the server where an instance runs, multiple instances may run concurrently, and if the VM running an instance fails then the system automatically restarts it.

Auto Scaling exploits cloud elasticity and provides automatic scaling of *EC2* instances. The service supports grouping of instances, monitoring of the instances in a group, and defining *triggers* and pairs of *CloudWatch alarms and policies*, which allow the size of the group to be scaled up or down. Typically, a maximum, a minimum, and a regular size for the group are specified. The *Elastic Beanstalk* service interacts with other *AWS* services, including *EC2, S3, SNS, Elastic Load Balance*, and *Auto Scaling*, automatically handles the deployment, capacity provisioning, load balancing, *Auto Scaling*, and application monitoring functions [83]. The service automatically scales the resources as required by the application, either up, or down based on default Auto Scaling settings. Some of the management functions provided by the service are: (i) deployment of a new application version (or rollback to a previous version); (ii) access to the results reported by *CloudWatch* monitoring service; (iii) email notifications when application status changes or application servers are added or removed; and (iv) access to server login files without needing to login to the application servers.

S3 is a storage service which allows an application to handle an unlimited number of objects ranging in size from one byte to five TB. An object is stored in a *bucket* and retrieved via a unique developer-assigned key. A bucket can be stored in a region selected by the user. *S3* maintains the name, modification time, an access control list, and up to four kilobytes of user-defined metadata for each object. The object names are global. Authentication mechanisms ensure that data is kept secure; objects can be made public, and rights can be granted to other users. The service ensures transparency of access, location, concurrency, replication, failure, migration, performance. Indeed, the user is not concerned with the physical location of the data, the data is automatically replicated and migrated. Multiple objects can be accessed concurrently. The same applies to the *Elastic Block Store (EBS)* which provides persistent

block-level storage volumes for use with Amazon *EC2* instances. A volume appears to an application as a raw, unformatted, and reliable physical disk.

Problem 2. *Compare the Oracle Cloud offerings (see https://cloud.oracle.com) with the cloud services provided by Amazon, Google, and Microsoft.*

Oracle supports both *IaaS* and *Paas* cloud delivery models and leverages its expertise in database management in its cloud service offerings. The utility computing supported by Oracle uses a different pricing model, a monthly subscription, rather than the *pay-as-you-go* model of AWS.

According to the site *http://www.oracle.com/us/solutions/cloud/infrastructure/overview* "Oracle Cloud Infrastructure provides a complete selection of servers, storage, networking fabric, virtualization software, operating systems, and management software to support diverse public and private cloud applications. Oracle engineers its hardware with application aware virtualization and management capabilities to enable the rapid deployment and efficient management of public and private IaaS."

Announced in January 2013, Oracle Infrastructure as a Service (Oracle IaaS) enables organizations to deploy fully integrated Engineered Systems, including Oracle Exadata Database Machine, Oracle Exalogic Elastic Cloud, Oracle SPARC SuperCluster, Oracle Exalytics In-Memory Machine and Oracle Sun ZFS Storage Appliance in their data centers behind their firewall, all for a simple monthly fee. The on-premise, private cloud infrastructure provides organizations with total control and visibility over their IT environments, enabling them to meet internal and regulatory compliance and security requirements.

Oracle IaaS includes elastic compute Capacity on Demand, enabling customers to easily add and remove processing capacity to meet their changing workloads, only paying for peak computing power when it is needed. Oracle IaaS includes Oracle Engineered Systems hardware. Exadata, Exalogic, and SPARC SuperCluster include Oracle Platinum Services and the new Oracle PlatinumPlus Services exclusively for Oracle IaaS customers. Oracle IaaS is part of the Oracle Private Cloud Services portfolio, which includes a comprehensive set of best-in-class integrated applications, platform and infrastructure products and solutions.

The Oracle Cloud Platform, a PaaS service, provides a shared and elastically scalable platform for consolidation of existing applications and new application development and deployment. It supports standards-based shared services and elastic scalability on demand and includes database functionality based on Oracle Database and Oracle Exadata Database Machine. It features middleware technology based on Oracle Fusion Middleware and Oracle Exalogic Elastic Cloud.

Problem 3. *Read the IBM report [41] and discuss the workload preferences for private and public clouds and the reasons for the preferences.*

The IBM study [41] asked customers to rate the 25 different workloads listed in Figure 4 they have already deployed or were planing to deploy on public and private clouds. Figure 5 summarizes the leading attractions to a public cloud. Figures 6 and 7 from [41] show the preferred private and public cloud workloads, respectively.

Problem 4. *In Section 3.7 we introduced the concept of energy-proportional systems and we saw that different system components have different dynamic ranges. Sketch a strategy to reduce the power consumption in a lightly-loaded cloud and discuss the steps for placing a computational server in a standby mode and then for bringing it back up to an active mode.*

Workload type	Workload
Analytics	• Data mining, text mining, or other analytics • Data warehouses or data marts • Transactional databases
Business services	• CRM or sales force automation • E-mail • ERP applications • Industry-specific applications
Collaboration	• Audio/video/Web conferencing • Unified communications • VoIP infrastructure
Desktop and devices	• Desktop
Development and test	• Development environment • Test environment
Infrastructure	• Application servers • Application streaming • Business continuity/disaster recovery • Data archiving • Data backup • Data center network capacity • Security • Servers • Storage • Training infrastructure • WAN capacity

Figure 4: Workloads discussed in the IBM survey.

We have seen in Section 3.7 that even at a very low load a server uses close to 50% of the energy consumption when delivering peak performance. The basic philosophy is to maintain all servers in an optimal region, a region where the relative performance is high while the relative power consumption is low. To achieve this goal, the applications running on a server working outside its optimal region should be migrated to other servers and the server should be switched to a sleep mode when its power consumption is negligible.

Problem 5. *Read the paper which introduced the concept of dataspaces [27] and analyze the benefits and the problems with this new idea. Research the literature for potential application of dataspaces to scientific data management in a domain of your choice, be it the search for the Higgs boson at CERN, structural biology, cancer research, or another important research topic which involves data-intensive applications.*

According to the paper "A New Abstraction for Information Management" *semantic inte-*

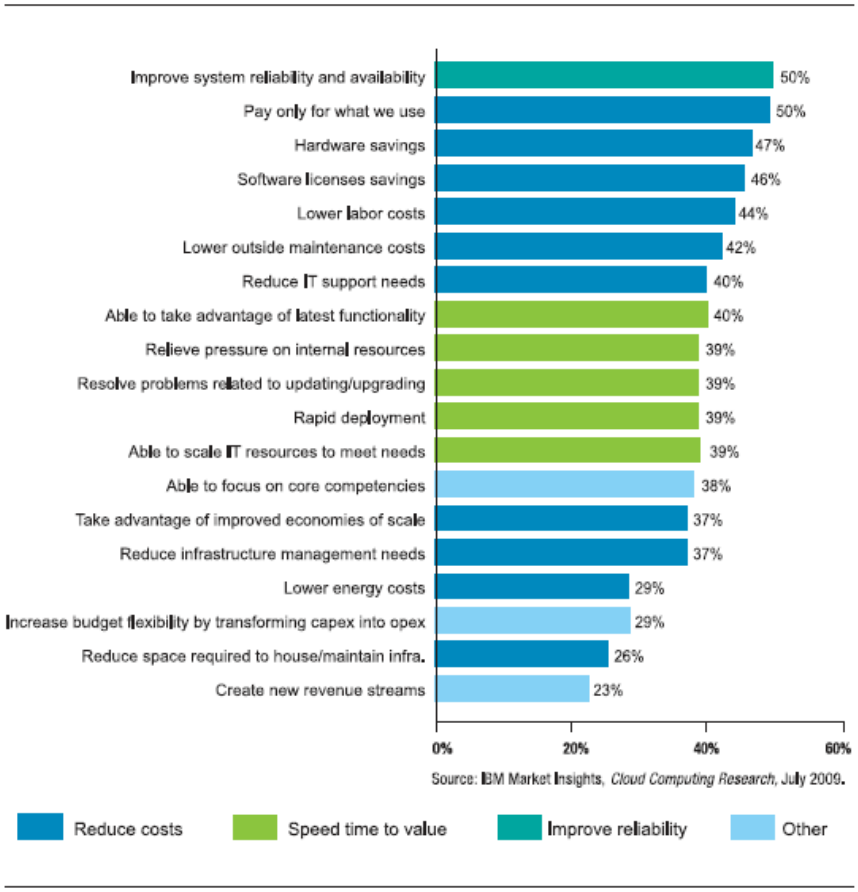

Figure 5: Public cloud drivers.

gration is a measure of how closely the schemas of the various data sources have been matched. In other words, how well the types, names, units, meanings, etc. of the data in the sources are matched up. At the high end of the spectrum, all data conforms to a single agreed-upon schema. At the low end, there is no schema information at all. In between lay various data integration solutions and approaches based on semi-structured data and controlled vocabularies. This dimension indicates the degree to which semantically rich query processing and data manipulation can be provided across a group of data sources, with higher degrees of integration providing richer functionality.

Data integration systems require semantic integration before any services can be provided. Hence, although there is not a single schema to which all the data conforms, the system knows the precise relationships between the terms used in each schema. As a result, a significant upfront effort is required in order to set up a data integration system.

Dataspaces are not a data integration approach; rather, they are more of a data co-existence approach. The goal of dataspace support is to provide base functionality over all data sources, regardless of how integrated they are. For example, a DSSP can provide keyword search over all of its data sources, similar to that provided by existing desktop search systems. When more sophisticated operations are required, such as relational-style queries, data mining, or monitoring over certain sources, then additional effort can be applied

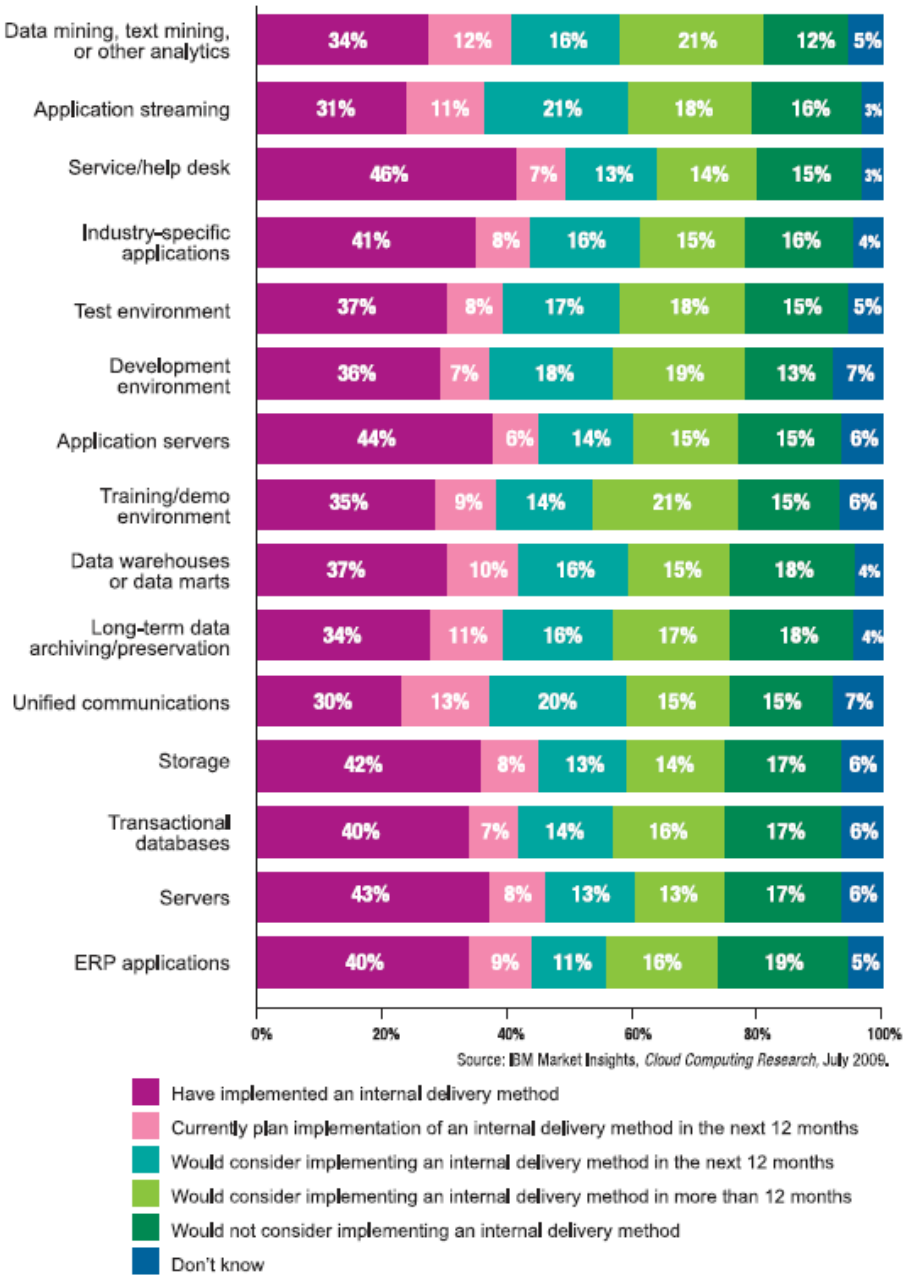

Figure 6: Private cloud workloads.

to more closely integrate those sources in an incremental, *pay-as-you-go* fashion.

Example: Consider a research group working on environmental observation and forecasting. They may be monitoring a coastal ecosystem through weather stations, shore and buoy-mounted sensors and remote imagery. In addition they can be running atmospheric and fluid-dynamics models that simulate past, current and near-future conditions. The computations may require importing data and model outputs from other groups, such as river flows and ocean circulation forecasts. The observations and simulations are the inputs to programs that generate a wide range of data products, for use within the group and by others: compar-

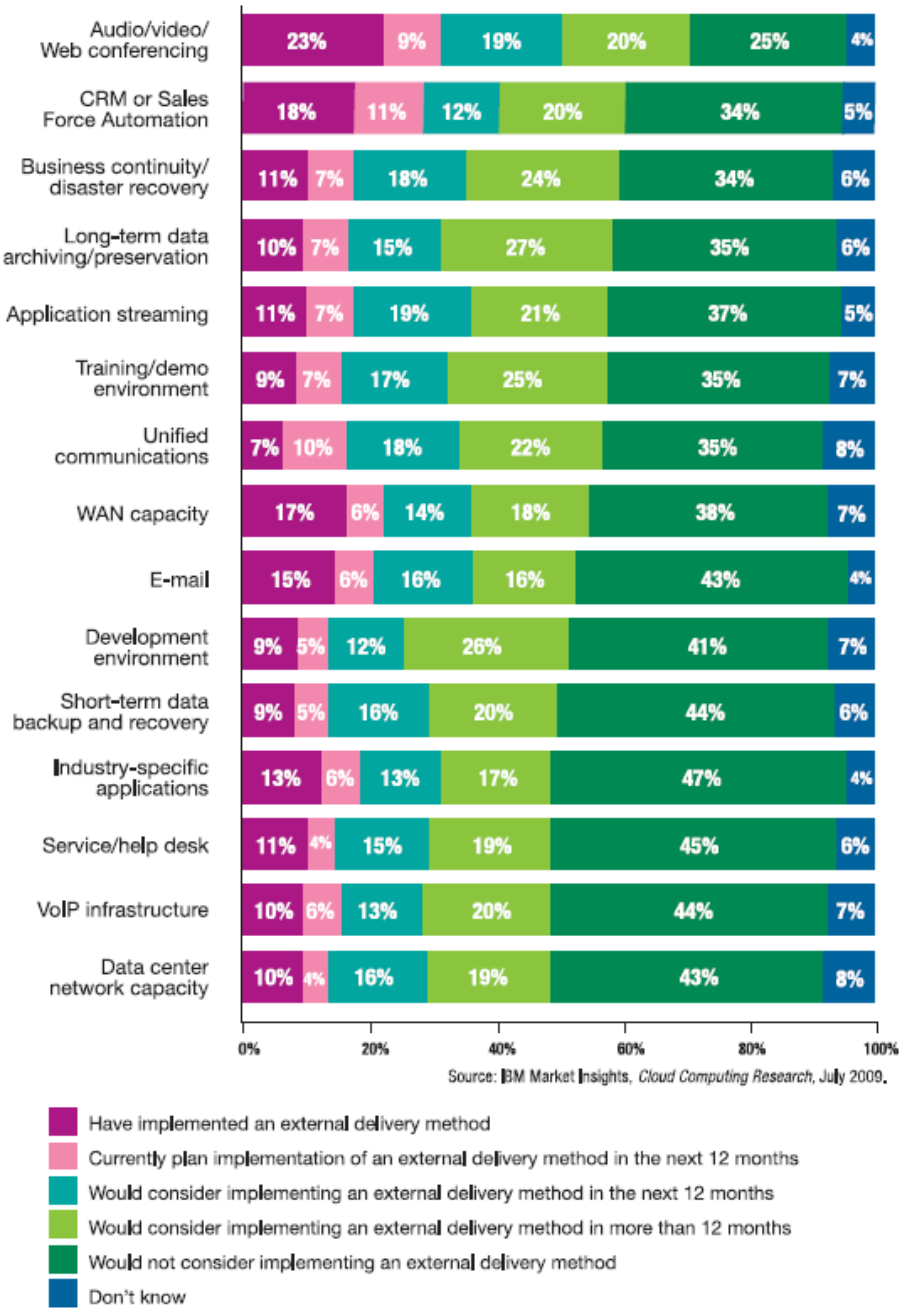

Figure 7: Public cloud workloads.

ison plots between observed and simulated data, images of surface temperature distributions, animations of salt-water intrusion into an estuary. Such a group can easily amass millions of data products in just a few years. While it may be that for each file, someone in the group knows where it is and what it means, no one person may know the entire holdings nor what every file means. People accessing this data, particularly from outside the group, would like to search a master inventory that had basic file attributes, such as time period covered, geographic region, height or depth, physical variable (salinity, temperature, wind speed), kind of

data product (graph, isoline plot, animation), forecast or hindcast, and so forth. Once data products of interest are located, understanding the lineage is paramount in being able to analyze and compare products: What code version was used? Which finite element grid? How long was the simulation time step? Which atmospheric dataset was used as input? Soon, such groups will need to federate with other groups to create scientific dataspaces of regional or national scope. They will need to easily export their data in standard scientific formats, and at granularities (sub-file or multiple file) that do not necessarily correspond to the partitions they use to store the data. Users of the federated dataspace may want to see collections of data that cut across the groups in the federation, such as all observations and data products related to water velocity, or all data related to a certain stretch of coastline for the past two months. Such collections may require local copies or additional indices for fast search. This scenario illustrates several dataspace requirements, including (1) a dataspace-wide catalog, (2) support for data lineage and (3) creating collections and indexes beyond what any one participating source supplies.

Problem 6. *In Section 3.7 it was mentioned that InfiniBand can be considered an energy proportional network. The network is used by supercomputers (see http://i.top500.org/); the InifiniBand fabric is also used to connect compute nodes, compute nodes with storage servers, and Exadata and Exalogic systems at Oracle data centers. Analyze the features of InfiniBand critical for reduction of energy consumption.*

InfiniBand is an interconnection network with a switched fabric topology designed to be scalable; it is used by supercomputers and computer clouds.It supports several signaling rates and the energy consumption depends on the throughput. Links can be bonded together for additional throughput; the *InfiniBand* architectural specification defines multiple operational data rates: single data rate (SDR), double data rate (DDR), quad data rate (QDR), fourteen data rate (FDR), and enhanced data rated (EDR). The signaling rates are: 2.5 Gbps in each direction per connection for an SDR connection; 5 Gbps for DDR; 10 Gbps for QDR; 14.0625 Gbps for FDR; and 25.78125 Gbps per lane for EDR . SDR, DDR, and QDR link encoding is 8 B/10 B, every 10 bits sent carry 8 bits of data. Thus single, double, and quad data rates carry $2, 4$, or 8 Gbps of useful data, respectively. The effective data transmission rate is four-fifths of the raw rate. *InfiniBand* allows links to be configured for a specified speed and width; the reactivation time of the link can vary from several nanoseconds to several microseconds. *InfiniBand* has high throughput and low latency and supports QoS guarantees and failover, the capability to switch to a redundant or standby system. It offers point-to-point bidirectional serial links intended for the connection of processors with high-speed peripherals, such as disks, as well as multicast operations.

Problem 7. *Many organizations operate one or more computer clusters and contemplate the migration to private clouds. What are the arguments for and against such an effort.*

The main arguments for such a move are reducing cost, and improving reliability, performance and usability. The maintenance costs are reduced as all systems use the same system software and software updates can be done automatically, thus lower labor costs. System reliability can be improved with the aid of a monitoring system like the *Cloud Watch* available on open cloud infrastructures such as *Eucalyptus.* Elasticity will allow applications to scale up and down thus, will improve system usability.

Problem 8. *Evaluate the SLA toolkit at http://www.service-level-agreement.net/. Is the interactive guide useful, what does it miss? Does the SLA template include all clauses that are important in your view, what is missing? Are the examples helpful?*

The site provides only general information. As pointed out in Section 3.8 the objective of an SLA are:

- Identify and define customers' needs and constraints, including the level of resources, security, timing, and quality of service.
- Provide a framework for understanding. A critical aspect of this framework is a clear definition of classes of service and costs.
- Simplify complex issues; for example, clarify the boundaries between the responsibilities of the clients and those of the provider of service in case of failures.
- Reduce areas of conflict.
- Encourage dialogue in the event of disputes.
- Eliminate unrealistic expectations.

An SLA records a common understanding in several areas: (i) services, (ii) priorities, (iii) responsibilities, (iv) guarantees, and (v) warranties. An agreement usually covers: services to be delivered, performance, tracking and reporting, problem management, legal compliance and resolution of disputes, customer duties and responsibilities, security, handling of confidential information, and termination.

Problem 9. *Software licensing is a major problem in cloud computing. Discuss several ideas to prevent an administrator from hijacking the authorization to use a software licence.*

The following discussion applies only to the *IaaS* model; indeed, in the *PaaS* model the SLA can specify the software a user has access to and prevent unauthorized use of licensed software; the system can maintain an access control list for each licensed software.

To verify the authorization to use a license, an application must have the certificate of an authority. This certificate must be available locally to the application because the application may be executed in an environment with restricted network access. This opens up the possibility for an administrator to hijack the license mechanism by exchanging the local certificate. The only feasible solution is to limit the number of concurrent use of the software license and, once the network access is restored, to inform the user granted the license of the activity during the past period.

This does not seem to be a realistic scenario as one of the premises of cloud computing is that the user has access to his computations running on the cloud. A possible solution is to keep a log of all license usage at the customer site and update and only allow the license to be activated after the log record was written.

Problem 10. *Annotation schemes are widely used by popular services such as Flickr photo-sharing service which supports annotation of photos. Sketch the organization of a cloud service used for sharing medical x-ray, tomography, CAT-scan and other medical images and discuss the main challenges for its implementation.*

The main challenge is providing a very strict privacy and security of the system. The system could be based on a client-server paradigm with a central image repository and clients running at the medical facilities that generate and use medical images. A new site could join the system following a very rigorous security check before being added to the system directory; a set of capabilities should be associated with every user at that site. A thorough procedure will allow a user from an already registered site to sign on. Every transaction to access an existing image will check the credentials of the requester, will require the explicit permission of the image owner and will be logged. An image will be stored in the database only after a record including the meta information related to the image is created. Comments added by individuals who had access to an image, and the log of all accesses to the image are included in the meta information record of each image. A possible solution is to use the Zookeeper to maintain a consistent view of the attributes of sites, individuals, and the meta information associated with all the objects in the database.

4 Chapter 4 - Applications

Problem 1. *Download and install the Zookeper from the site http://zookeeper.apache.org/. Use the API to create the basic workflow patterns in Figure 8.*

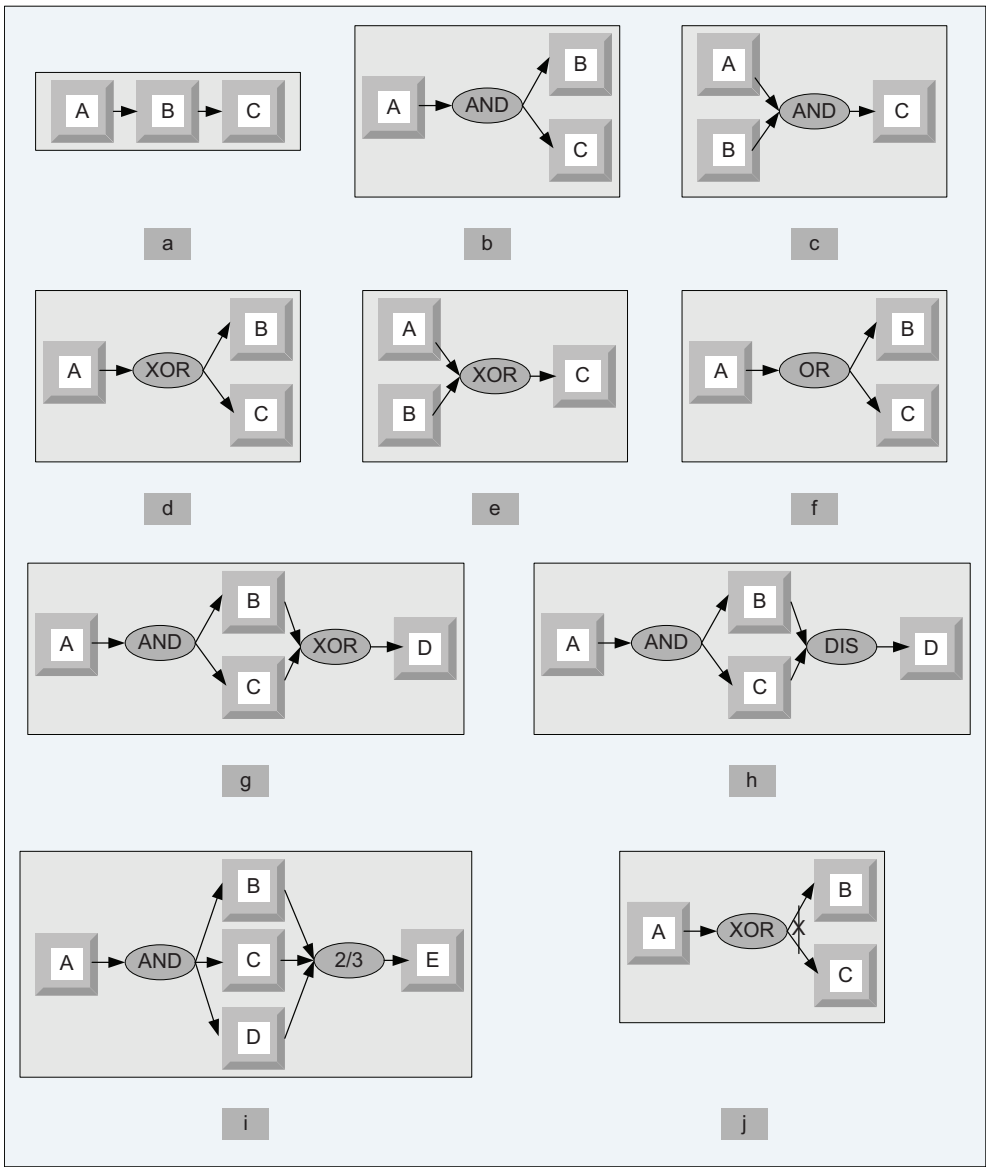

Figure 8: Basic workflow patterns. (a) Sequence; (b) AND split; (c) Synchronization; (d) XOR split; (e) XOR merge; (f) OR split; (g) Multiple Merge; (h) Discriminator; (i) N out of M join; (j) Deferred Choice.

```
Standalone Operation
Setting up a ZooKeeper server in standalone mode is straightforward.
The server is contained in a single JAR file, so installation consists
of creating a configuration.
```

Once you've downloaded a stable ZooKeeper release unpack it and cd to the root

To start ZooKeeper you need a configuration file. Here is a sample,
create it in conf/zoo.cfg:

tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
This file can be called anything, but for the sake of this discussion call
it conf/zoo.cfg. Change the value of dataDir to specify an existing (empty to
start with) directory. Here are the meanings for each of the fields:

tickTime
the basic time unit in milliseconds used by ZooKeeper. It is used to do
heartbeats and the minimum session timeout will be twice the tickTime.

dataDir
the location to store the in-memory database snapshots and, unless
specified otherwise, the transaction log of updates to the database.

clientPort
the port to listen for client connections

Now that you created the configuration file, you can start ZooKeeper:

bin/zkServer.sh start
ZooKeeper logs messages using log4j -- more detail available in
the Logging section of the Programmer's Guide. You will see log
messages coming to the console (default) and/or a log file depending
on the log4j configuration.

The steps outlined here run ZooKeeper in standalone mode. There is
no replication, so if ZooKeeper process fails, the service will go down.
This is fine for most development situations, but to run ZooKeeper in
replicated mode, please see Running Replicated ZooKeeper.

Managing ZooKeeper Storage
For long running production systems ZooKeeper storage must be managed
externally (dataDir and logs). See the section on maintenance for more details.

Connecting to ZooKeeper
Once ZooKeeper is running, you have several options for connection to it:

Java: Use
bin/zkCli.sh -server 127.0.0.1:2181
This lets you perform simple, file-like operations.
C: compile cli_mt (multi-threaded) or cli_st (single-threaded) by running

make cli_mt or make cli_st in the src/c subdirectory in the ZooKeeper sources.
See the README contained within src/c for full details.
You can run the program from src/c using:
LD_LIBRARY_PATH=. cli_mt 127.0.0.1:2181
or
LD_LIBRARY_PATH=. cli_st 127.0.0.1:2181
This will give you a simple shell to execute file system like operations
on ZooKeeper. Once you have connected, you should see something like:

```
Connecting to localhost:2181
log4j:WARN No appenders could be found for logger (org.apache.zookeeper.ZooKeeper).
log4j:WARN Please initialize the log4j system properly.
Welcome to ZooKeeper!
JLine support is enabled
[zkshell: 0]
```

From the shell, type help to get a listing of commands that can be executed from
the client, as in:

```
[zkshell: 0] help
ZooKeeper host:port cmd args
        get path [watch]
        ls path [watch]
        set path data [version]
        delquota [-n|-b] path
        quit
        printwatches on|off
        createpath data acl
        stat path [watch]
        listquota path
        history
        setAcl path acl
        getAcl path
        sync path
        redo cmdno
        addauth scheme auth
        delete path [version]
        deleteall path
        setquota -n|-b val path
```

From here, you can try a few simple commands to get a feel for this
simple command line interface. First, start by issuing the list command,
as in ls, yielding:

```
[zkshell: 8] ls /
[zookeeper]
```

Next, create a new znode by running create /zk_test my_data. This creates a new znode and associates the string "my_data" with the node. You should see:

```
[zkshell: 9] create /zk_test my_data
Created /zk_test
```

Issue another ls / command to see what the directory looks like:

```
[zkshell: 11] ls /
[zookeeper, zk_test]
```

Notice that the zk_test directory has now been created.

Next, verify that the data was associated with the znode by running the get command, as in:

```
[zkshell: 12] get /zk_test
my_data
cZxid = 5
ctime = Fri Jun 05 13:57:06 PDT 2009
mZxid = 5
mtime = Fri Jun 05 13:57:06 PDT 2009
pZxid = 5
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0
dataLength = 7
numChildren = 0
```

We can change the data associated with zk_test by issuing the set command, as in:

```
[zkshell: 14] set /zk_test junk
cZxid = 5
ctime = Fri Jun 05 13:57:06 PDT 2009
mZxid = 6
mtime = Fri Jun 05 14:01:52 PDT 2009
```

```
pZxid = 5
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0
dataLength = 4
numChildren = 0
[zkshell: 15] get /zk_test
junk
cZxid = 5
ctime = Fri Jun 05 13:57:06 PDT 2009
mZxid = 6
mtime = Fri Jun 05 14:01:52 PDT 2009
pZxid = 5
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0
dataLength = 4
numChildren = 0

(Notice we did a get after setting the data and it did, indeed, change.

Finally, let's delete the node by issuing:


[zkshell: 16] delete /zk_test
[zkshell: 17] ls /
[zookeeper]
[zkshell: 18]
```

Problem 2. *Use the AWS Simple Workflow Service to create the basic workflow patterns in Figure 8.*

The *Elastic Beanstalk* provides a better environment than the *simple Workflow Service*. To use it:

```
1. Sign in to the AWS account.
2. Select the "AWS Management Console" from the left panel.
3. Select the "AWS Elastic Beanstalk" under "Deployment and Management"
4. Configure the AWS settings
```

Problem 3. *Use the AWS CloudFormation service to create the basic workflow patterns in Figure 8.*

```
1. Sign in to the AWS account.
2. Select the "AWS Management Console" from the left panel.
3. Select the "AWS Cloud Formation" under "Deployment and Management"
4. Describe the AWS resources needed to run the application in a simple text
   file called a template and AWS CloudFormation takes care of provisioning.
5. A good way to implement several workflow patterns is to use SNS
   (Simple Notification Service). To do so:
   5.1 Use a UI to create a new topic
   5.2 Manage the subscribers
   5.3 Publish messages
   5.4 Set policies, configure access control.
```

Problem 6. *The paper [15] describes the elasticLM, a commercial product which provides license and billing Web-based services. Analyze the merits and the shortcomings of the system.*

The *elasticLM* is a technology for creating and managing software licenses designed for distributed computing environments like Clouds. License usage is authorized through negotiation of Service Level Agreements between the license service and the user taking into account availability, policies defined locally, policies of the service provider or attributes defined for user in a virtual organization. The negotiation allows to inform the user about the price of the license before executing the application. The price information can also be used to check the request against budget constraints defined, e.g., for user or department.

The result of a successful negotiation is a mobile token which can be used e.g. in a *IaaS* environment, to execute an application with the agreed features. Tokens can move to the environment where needed, can be combined with other tokens or licenses created on the fly by an ASP.

Software licenses are treated and implemented as services, thus providing platform independent access just like any other visualized resources. Licenses as services overcome the limitations of current monolithic licensing models. Licenses will be managed as agreements, extending the conventional Service Level Agreements (SLAs) which are made today between sellers and buyers in the market. Licenses will be dynamic in order to support agreements that may change over time and where the dynamic negotiation between service provider and consumer is needed. In addition to licensing and license management, *elasticLM* offers additional capabilities like monitoring the use of applications (license protected or not) and creating usage records for later evaluation.

Moreover, *elasticLM* may also be used in *SaaS* or *PaaS* environments - e.g., where the license is available locally and no tokens are needed for application execution - to create Service Level Agreements with the *SaaS* user and create usage records for accounting and billing.

Special care is paid to security. Issues related to: authentication and authorization of users, services, and servers; security and confidentiality of the communication between different actors or component;, security of the delegation process, when using a portal or an orchestrator, for example; disclosure of sensitive information, e.g. compromise of licenses integrity of the process to inhibit non-repudiation, are part of the process.

Problem 7. *Search the web for reports of cloud system failures and discuss the causes of each incident.*

Clouds are affected by malicious attacks and failures of the infrastructure e.g., power failures. Such events can affect Internet domain name servers and prevent access to a cloud or can directly affect the clouds. For example, an attack at Akamai on June 15, 2004 caused a domain name outage and a major blackout that affected Google, Yahoo!, and many other sites. In May 2009 Google was the target of a serious denial-of-service (DoS) attack that took down services such Google News and Gmail for several days.

Lightning caused a prolonged downtime at Amazon on June 29 and 30, 2012; the *AWS* cloud in the Eastern region of the United States, which consists of 10 data centers across four availability zones, was initially troubled by utility power fluctuations, probably caused by an electrical storm. A June 29, 2012 storm on the East Coast took down some Virginia-based Amazon facilities and affected companies using systems exclusively in this region. *Instagram*, a photo-sharing service, was one of the victims of this outage. The recovery from the failure took a very long time and exposed a range of problems. For example, one of the 10 centers failed to switch to backup generators before exhausting the power that could be supplied by *uninterruptible power supply* (UPS) units. *AWS* uses "control planes" to allow users to switch to resources in a different region, and this software component also failed. The booting process was faulty and extended the time to restart *EC2* (*Elastic Computing*) and *EBS* (*Elastic Block Store*) services. Another critical problem was a bug in the *Elastic Load Balancer (ELB)*, which is used to route traffic to servers with available capacity. A similar bug affected the recovery process of the Relational Database Service (RDS). This event brought to light "hidden" problems that occur only under special circumstances.

Problem 8. *Identify a set of requirements you would like to be included in a Service Level Agreement. Attempt to express them using the Web Service Agreement Specification (WS-Agreement) [6] and determine if it is flexible enough to express your options.*

An SLA records a common understanding in several areas: (i) services, (ii) priorities, (iii) responsibilities, (iv) guarantees, and (v) warranties. An agreement usually covers: services to be delivered, performance, tracking and reporting, problem management, legal compliance and resolution of disputes, customer duties and responsibilities, security, handling of confidential information, and termination.

WSLA consists of a set of concepts and a XML language. It is designed to capture service level agreements in a formal way. WSLA comprises three entities [56]:

- Parties. WSLA contains the descriptions of: service provider; service consumer; and third parties. The task of these third parties may vary from measuring service parameters to taking actions on violations as delegated by either the service provider or the service consumer.

- SLA parameters specified by metrics. Metrics define how service parameters can be measured and are typically functions. There are at least two major types of metrics:

  - Resource metrics are retrieved directly from the provider resources and are used as is without further processing. For example, transaction count.
  - Composite metrics represents a combination of several resource metrics, calculated according to a specific algorithm. For example, transactions per hour combines the raw resource metrics of transaction count and uptime. Composite metrics are required when the consumers need insightful and contextual information where raw

numbers do not suffice. A third metrics referred to as a business metric has been defined in [2]. It relates SLA parameters to financial terms specific to a service customer.

- Service Level Objectives(SLOs) expressed as a set of formal expressions. These formal expressions have the *if...then* structure. The antecedent (if) contains conditions and the consequent (then) contains actions. An action represents what a party has agreed to perform when the conditions are met.

An established WSLA contains the following major sections:

1. Parties. This section comprises of two parties : supporting parties and signatory parties. Signatory parties are the service provider and the service consumer. Supporting parties are the third parties that come into picture when signatory parties decide to delegate certain tasks such as measuring SLA parameters.

2. Service Definitions. A service definition contains the description of the service providers interface. Services are represented by service objects. Each service object associates with one or more SLA parameters.

3. Obligations. This section contains the conditions and the action guarantees.

Problem 9. *Research the power consumption of processors used in mobile devices and their energy efficiency. Rank the components of a mobile device in terms of power consumption. Establish a set of guidelines to minimize the power consumption of mobile applications.*

First, we overview the mobile processors produced by several manufacturers and the mobile systems using them. The ARM architecture was developed by ARM Holdings which does not manufacture its own electronic chips, but licenses its designs to other semiconductor manufacturers. The name ARM is an acronym for Advanced RISC Machine.

The ARM architecture is the primary hardware environment for most mobile device operating systems such as iOS, Android, Windows Phone, Blackberry OS/Blackberry 10, Firefox OS, Tizen and Ubuntu Touch. ARM-based processors and *systems-on-a-chip* include the Qualcomm Snapdragon, nVidia Tegra, Marvell Xscale and Texas Instruments OMAP, as well as ARM's Cortex series and Apple System on Chips (used in its iPhones).

Nvidia's Tegra 3 processor has been the first successful quad-core mobile processor. The Tegra 3 has what the company terms a 12-core GeForce GPU and new video engines with support of 1080p video at 40Mbps. The processor is used in HTC One X, LG Optimus 4X HD, and ZTE Era, as well as a new high-end phone from Fujitsu and some new tablets from Asus. Nvidia's 4-Plus-1 architecture has four high-power A9s and one low-power one.

Qualcomm's processors are in a large range of products, from some models of the Samsung Galaxy Note and Galaxy S II to the HTC One S, XL, and virtually all Windows Phones. Snapdragon S4 MSM8960 is a dual-core chip based on the company's new "Krait" architecture, which seems to be the first 28nm mobile processor shipping in volume. Qualcomm is designing its own cores that are compatible with the ARM v7 instruction set, rather than using the standard ARM cores. The company also has its own graphics engine, known as Adreno. What really makes this stand out is an integrated LTE modem.

Texas Instruments uses the OMAP 4 family of processors in Kindle Fires and Nook tablets, as well as the first Android 4.0 devices like the Google Nexus. The upcoming OMAP 5 family

will introduce the new ARM Cortex-A15 core, faster and more powerful, with twin SGX-544 graphics cores, video compression, and two Cortex-M4s. In this "smart multicore architecture," the M4 processors handle a real-time embedded OS, while the A15s run Android.

Intel Core i7 Extreme Edition has a maximum turbo frequency of 3.5 GHz, 8 MB as L3 cache, and is realized with the 32 nm technology; the memory speed is 1.6 GHz.

There is very little information available about power consumption of mobile processors. Table 2 shows the evolution of the average power consumption for volume (Vol) servers - servers with a price less than $ 25 K, mid-range (Mid) servers - servers with a price between $25 K and $499 K, and high-end (High) servers - servers with a price tag larger than $500 K.

Table 2: Estimated average power use of volume, mid-range, and high-end servers (in Watts) along the years.

Type	2000	2001	2002	2003	2004	2005	2006
Vol	186	193	200	207	213	219	225
Mid	424	457	491	524	574	625	675
High	5,534	5,832	6,130	6,428	6,973	7,651	8,163

The largest consumer of power of a system is the processor, followed by memory, and storage systems. The power consumption can vary from 45W to 200W per multi-core CPU; newer processors include power saving technologies. Most servers have several dual in-line memory modules (DIMMs); the power consumption of and a DIMM can vary from 5W up to 21W. Large servers often use 32 to 64 DIMMs; server memory cooling requires additional power. A server with 2-4 hard disk drives (HDDs) consumes 24 - 48W.

5 Chapter 5 - Virtualization

Problem 1. *Identify the milestones in the evolution of operating systems during the half century from* 1960 *to* 2010 *and comment on the statement from [68] "VMMs give operating system developers another opportunity to develop functionality no longer practical in today's complex and ossified operating systems, where innovation moves at a geologic pace."*

IBM System/360. In early 1960s IBM started the development of System/360. S/360 would span an unprecedented range of processing power, memory size, device support, and cost; and more important, it was based on a pledge of backward compatibility, such that any customer could move software to a new system without modification. In today's world of standard interfaces and portable systems, this may not seem such a radical goal; but at the time, it was revolutionary. Before the S/360, each computer model often had its own specific devices that could not be used with other systems. Buying a bigger CPU also meant buying new printers, card readers, tape drives, etc. With the S/360, IBM wanted to offer a huge range of computer systems, all sharing a single processor architecture, instruction set, I/O interface, and operating system. Customers would be able to "mix and match" to meet current needs; and they could confidently upgrade their systems in the future, without the need to rewrite all their software applications. IBM's focus remained on its traditional customer base: large organizations doing administrative and business data processing. IBM's S/360 announcement in April 1964 did not include key elements, particularly virtual memory capabilities.

Multics (Multiplexed Information and Computing Service) is a mainframe timesharing operating system which was designed around 1965 and used until 2000. It began as a research project and was an important influence on operating system development. It also pioneered the use of high-level languages for writing operating systems. It also was one of the first OS which paid serious attention o the security. History has shown that it was a real pioneer which introduced concepts now taken for granted. PL/1 was used as system programming language, paradoxically it was also at least 20 years ahead of its time; many key ideas such as exceptions handing, built-in strings, etc were "reinvented" on C++ and Java many years after they were introduced in PL/1. Multics also was a pioneer in computer security, being essentially an opposite of Unix. Many of the security innovations in Multics found their way in Unix only 30 years later.

Unix is a multitasking, multi-user computer operating system originally developed in 1969 at Bell Labs, by Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy, Michael Lesk and Joe Ossanna. During the late 1970s and early 1980s, the influence of Unix in academic circles led to large-scale adoption of Unix (particularly of the BSD variant, originating from the University of California, Berkeley) by commercial startups, the most notable of which are Solaris, HP-UX, Sequent, and AIX, as well as Darwin, which forms the core set of components upon which Apple's OS X and iOS are based. Today, in addition to certified Unix systems such as those already mentioned, Unix-like operating systems such as MINIX, Linux, and BSD descendants (FreeBSD, NetBSD, OpenBSD, and DragonFly BSD) are commonly encountered. The term *traditional Unix* is used to describe an operating system that has the characteristics of either Version 7 Unix or UNIX System V.

Linux was originally developed as a free operating system for Intel x86-based personal computers. The defining component of Linux is the Linux kernel, an operating system kernel first released on 5 October 1991, by Linus Torvalds. It has since been ported to more computer hardware platforms than any other operating system. It is a leading operating system on

servers and other big iron systems such as mainframe computers and supercomputers - more than 90% of today's 500 fastest supercomputers run some variant of Linux, including the 10 fastest. Linux also runs on embedded systems (devices where the operating system is typically built into the firmware and highly tailored to the system) such as mobile phones, tablet computers, network routers, building automation controls, televisions and video game consoles; the Android system, in wide use on mobile devices, is built on the Linux kernel. The development of Linux is one of the most prominent examples of free and open source software collaboration: the underlying source code may be used, modified, and distributedcommercially or non-commercially by anyone under licenses such as the GNU General Public License. Typically, Linux is packaged in a format known as a Linux distribution for desktop and server use. Some popular mainstream Linux distributions include Debian (and its derivatives such as Ubuntu and Linux Mint), Red Hat Enterprise Linux (and its derivatives such as Fedora and CentOS), Mandriva/Mageia, openSUSE (and its commercial derivative SUSE Linux Enterprise Server), and Arch Linux. Linux distributions include the Linux kernel, supporting utilities and libraries and usually a large amount of application software to fulfill the distribution's intended use.

The operating systems of the 1980's were: the DOS family, SUN OS, GNU, LISA OS, and Windows. The 1990's saw the deployment of the Windows family, Linux, and MAC OS. The Windows XP, Vista, and System 7, from Microsoft, OS X and iOS from Apple and Linux dominate the 2000's. iOS is used by iPhone and iPAD. Linux is a free and secure OS used by PCs, supercomputers and workstations; there are over 600 distributions with 5 main bases, Debian, Gentoo, Pacman, RPM, Slackware. The Android is an OS based on Linux used in many mobile devices; its development is lead by Google.

Problem 2. *Virtualization simplifies the use of resources, isolates users from one another, supports replication and mobility, but exacts a price in terms of performance and cost. Analyze each one of these aspects for: (i) memory virtualization, (ii) processor virtualization, and (iii) virtualization of a communication channel.*

Memory virtualization. It does simplify the use of systems as it allows an application to run on systems with different sizes of physical memory. It confines a user to its own virtual address space and the DAT (Dynamic Address Translation) helps isolate processes from one another. It can affect the performance when the working set of a process is larger than the physical memory available. It increases slightly the cost of the hardware, the complexity of the kernel, and it adds to the difficulties for processor virtualization.

Processor virtualization. It is critical for concurrency, a thread being a virtual processor. It is also critical for cloud computing, in particular for the *IaaS* paradigm; it allows multiple operating systems and applications to run on the same server. The price to pay is the overhead for context switching in the case of multi-threading and the overhead introduced by a hypervisor in the case of VMs.

Communication channel virtualization. Communication protocols support flow control, error control, and congestion control thus, they simplify the development of applications. Applications end up operating on virtual bit pipes with desirable properties, e.g., error-free.

Problem 3. *Virtualization of the processor combined with virtual memory management pose multiple challenges; analyze the interaction of interrupt handling and paging.*

Context switching could involve multiple components of a OS kernel including the Virtual Memory Manager (VMM), the Exception Handler (EH), the Scheduler (S), and the Multi-level Memory Manager (MLMM). When a page fault occurs during the fetching of the next instruction, multiple context switches are necessary as shown in Figure 9.

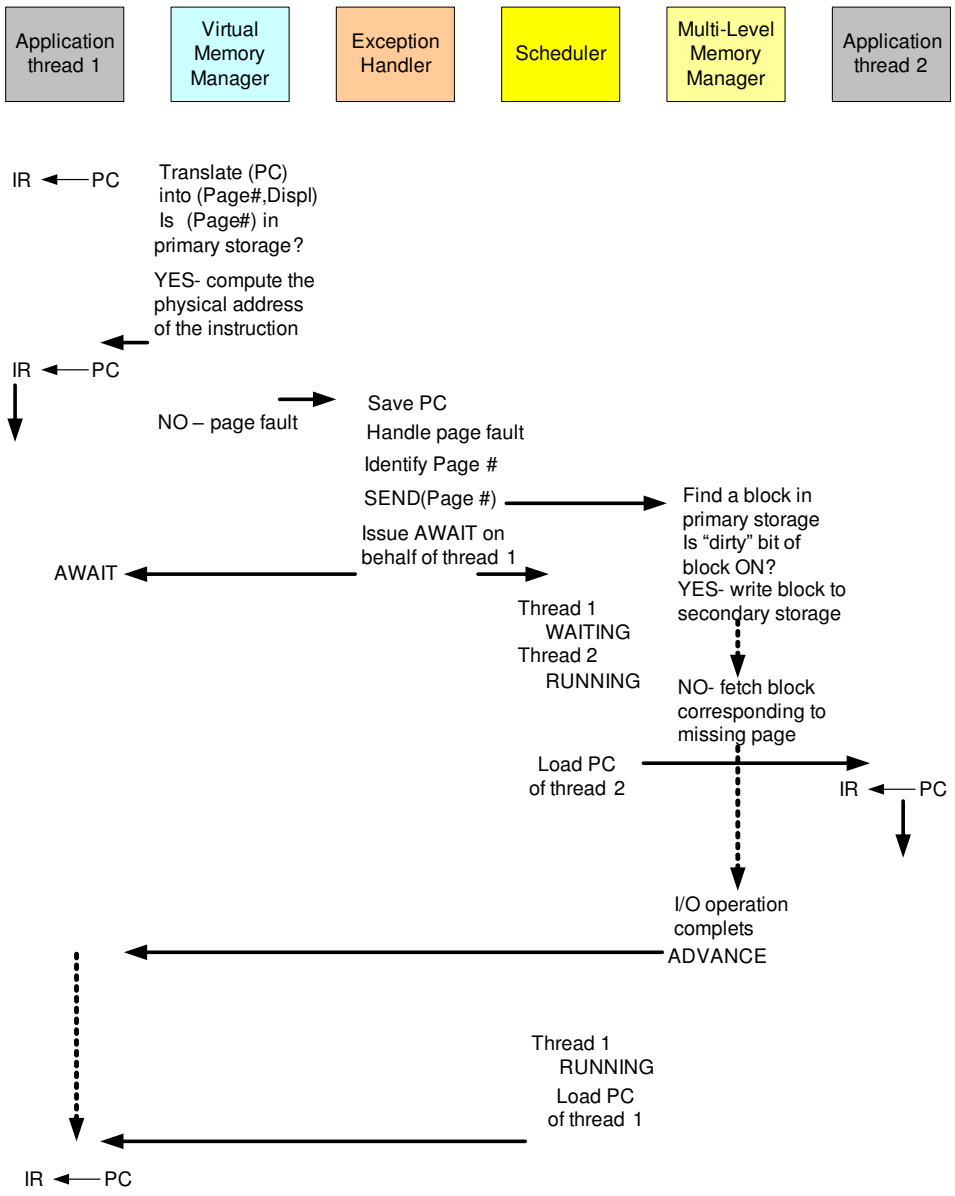

Figure 9: Context switching when a page fault occurs during the instruction fetch phase. VMM attempts to translate the virtual address of a next instruction of thread 1 and encounters a page fault. Then thread 1 is suspended waiting for an event when the page is brought in the physical memory from the disk. The Scheduler dispatches thread 2. To handle the fault the EX invokes the MLMM.

On a pipelined processor a page fault could occur at any time and special precautions must be taken to guarantee state consistency.

Problem 4. *In Section 5.5 we stated that a VMM is a much simpler and better specified system than a traditional operating system. The security vulnerability of VMMs is considerably reduced, as the systems expose a much smaller number of privileged functions. Research the literature to gather arguments in support of these affirmations; compare the number of lines of code and of system calls for several operating systems including Linux, Solaris, FreeBSD, Ubuntu, AIX, and Windows with the corresponding figures for several system virtual machines in Table 3.*

Table 3: A non-exhaustive inventory of system virtual machines. The host ISA refers to the instruction set of the hardware; the guest ISA refers to the instruction set supported by the virtual machine. The VM could run under a host OS, directly on the hardware, or under a VMM. The guest OS is the operating system running under the control of a VM which, in turn, may run under the control of the virtual machine monitor.

Name	Host ISA	Guest ISA	Host OS	guest OS	Company
Integrity VM	*x86*-64	*x86*-64	HP-Unix	Linux,Windows HP Unix	HP
Power VM	Power	Power	No host OS	Linux, AIX	IBM
z/VM	z-ISA	z-ISA	No host OS	Linux on z-ISA	IBM
Lynx Secure	*x86*	*x86*	No host OS	Linux, Windows	LinuxWorks
Hyper-V Server	*x86*-64	*x86*-64	Windows	Windows	Microsoft
Oracle VM	*x86*, *x86*-64	*x86*, *x86*-64	No host OS	Linux, Windows	Oracle
RTS Hypervisor	*x86*	*x86*	No host OS	Linux, Windows	Real Time Systems
SUN xVM	*x86*, SPARC	same as host	No host OS	Linux, Windows	SUN
VMware EX Server	*x86*, *x86*-64	*x86*, *x86*-64	No host OS	Linux, Windows Solaris, FreeBSD	VMware
VMware Fusion	*x86*, *x86*-64	*x86*, *x86*-64	MAC OS *x86*	Linux, Windows Solaris, FreeBSD	VMware
VMware Server	*x86*, *x86*-64	*x86*, *x86*-64	Linux, Windows	Linux, Windows Solaris, FreeBSD	VMware
VMware Workstation	*x86*, *x86*-64	*x86*, *x86*-64	Linux, Windows	Linux, Windows Solaris, FreeBSD	VMware
VMware Player	*x86*, *x86*-64	*x86*, *x86*-64	Linux Windows	Linux, Windows Solaris, FreeBSD	VMware
Denali	*x86*	*x86*	Denali	ILVACO, NetBSD	University of Washington
Xen	*x86*, *x86*-64	*x86*, *x86*-64	Linux Solaris	Linux, Solaris NetBSD	University of Cambridge

The Source Lines of Code (SLOC) is a measure of the size of a software project including an operating system. The evolution in time of this measure for Microsoft products in million lines of code: 4-5 for Windows NT 3.1 released in 1993, 7-8 for Windows NT 3.5 released in 1994, 11-12 for Windows NT 4.0 released in 1996, more than 29 for Windows 2000, 45 for

Windows XP released in 2001, and 50 for Windows server 2003.

The Linux Foundation celebrated the kernel's 20th birthday in 2012, alongside the release of Linux 3.0. The total size of the kernel grew from 13 million lines of code and 33,000 files in 2010 to 15 million lines of code and 37,000 files in 2011.

The SLOC for other operating systems:86 for Mac OS X 10.4; 324 for Debian 5.0; 9.7 for Open Solaris; and 8.8 for FreeBSD.

A VMM is a much simpler and better specified system than a traditional operating system; for example, the *Xen* VMM has approximately 60 000 lines of code while the *Denali* VMM [86] has only about half, 30 000 lines of code.

Problem 5. *In Section 5.6 we state that a VMM for a processor can be constructed if the set of sensitive instructions is a subset of the privileged instructions of that processor. Identify the set of sensitive instructions for the x86 architecture and discuss the problem each one of these instruction poses.*

The problems faced by virtualization of the *x86* architecture:

- *Ring deprivileging;* this means that a VMM forces the guest software, the operating system and the applications to run at a privilege level greater than 0. Recall that the *x86* architecture provides four protection rings, 0-3. Two solutions are then possible: (a) the (0/1/3) mode when the VMM, the OS, and the application run at privilege levels $0, 1$ and 3, respectively; (b) the $(0, 3, 3)$ mode when the VMM, a guest OS, and applications run at privilege levels $0, 3$ and 3, respectively. The first mode is not feasible for *x86* processors in 64-bit mode, as we shall see shortly.

- *Ring aliasing*; problems are created when a guest OS is forced to run at a privilege level other than that it was originally designed for. For example, when the CS register[2] is PUSHed, the current privilege level in the CR is also stored on the stack [62].

- *Address space compression;* a VMM uses parts of the guest address space to store several system data structures such as the interrupt-descriptor table and the global-descriptor table. Such data structures must be protected, but the guest software must have access to them.

- *Non-faulting access to privileged state;* several instructions `LGDT, SIDT, SLDT` and `LTR` loading the registers `GDTR, IDTR, LDTR`, and `TR` can only be executed by software running at privileged level 0 because these instructions point to data structures that control the CPU operation. Nevertheless, instructions that store from these registers fail silently when executed at a privilege level other than 0. This implies that a guest OS executing one of these instructions does not realize that the instruction has failed.

- *Guest system calls*; two instructions `SYSENTER` and `SYSEXIT` support low-latency system calls. The first causes a transition to privilege level 0, while the second causes a transition from privilege level 0 and fails if executed at a level higher than 0. The VMM must then emulate every guest execution of either of these instructions and this has a negative impact on performance.

[2]The *x86* architecture supports memory segmentation with a segment size of 64K. The CR (code-segment register) points to the code segment. MOV, POP, and PUSH instructions serve to load and store segment registers, including CR.

- *Interrupt virtualization;* in response to a physical interrupt the VMM generates a "virtual interrupt" and delivers it later to the target guest OS. But every OS has the ability to mask interrupts thus, the virtual interrupt could only be delivered to the guest OS when the interrupt is not masked. Keeping track of all guest OS attempts to mask interrupts greatly complicates the VMM and increases the overhead.

- *Access to hidden states;* elements of the system state, e.g., descriptor caches for segment registers, are hidden. There is no mechanism for saving and restoring the hidden components when there is a context switch from one VM to another.

- *Ring compression;* paging and segmentation are the two mechanisms to protect VMM code from being overwritten by guest OS and applications. Systems running in 64-bit mode can only use paging, but paging does not distinguish between privilege levels 0, 1, and 2, thus the guest OS must run at privilege level 3, the so called (0/3/3) mode. Privilege levels 1 and 2 cannot be used thus, the name *ring compression.*

- *Frequent access to privileged resources increases VMM overhead;* the task-priority register (`TPR`) is frequently used by a guest OS. The VMM must protect the access to this register and trap all attempts to access it, this can cause a significant performance degradation.

Problem 6. *Table 4 summarizes the effects of Xen network performance optimization reported in [60]. The send data rate of a guest domain is improved by a factor of more than* 4*, while the improvement of the receive data rate is very modest. Identify several possible reasons for this discrepancy.*

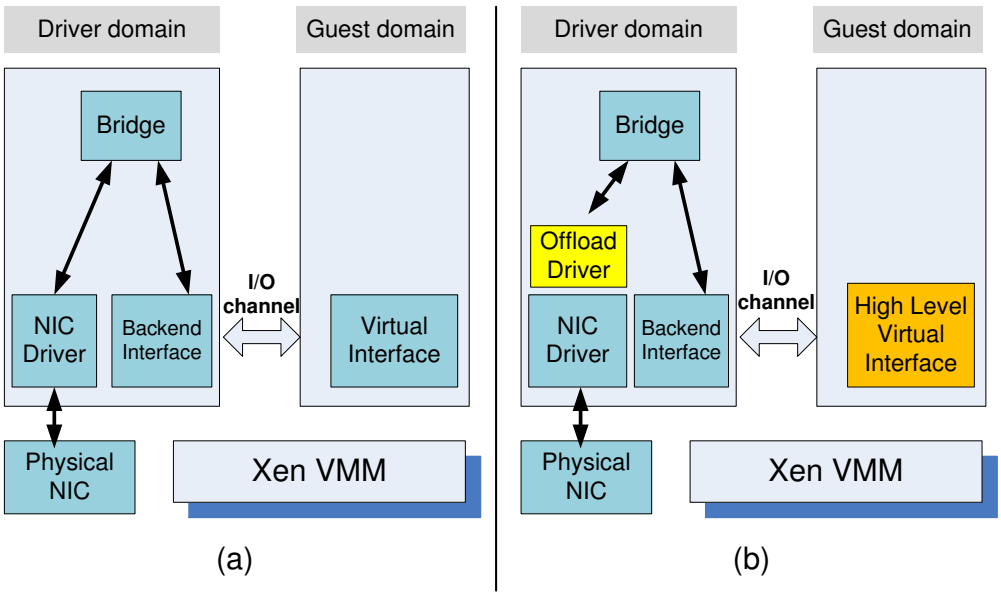

Figure 10: *Xen* network architecture (a) The original architecture; (b) The optimized architecture.

Figure 10 (a) shows the basic network architecture of *Xen*. Privileged operations, including I/O, are executed by *Dom0* on behalf of a guest operating system; in this context we shall refer to it as the *driver domain* called in to execute networking operations on behalf of the *guest domain*. The *driver domain* uses the native Linux driver for the NIC (Network Interface Controller) which in turn communicates with the physical NIC, also called the network adapter. The *guest domain* communicates with the *driver domain* through an I/O channel; more precisely, the guest OS in the guest domain uses a virtual interface to send/receive data to/from the backend interface in the driver domain.

Table 4: A comparison of send and receive data rates for a native `Linux` system, the *Xen* driver domain, an original *Xen* guest domain, and an optimized *Xen* guest domain.

System	Receive data rate (Mbps)	Send data rate (Mbps)
Linux	2 508	3 760
Xen driver	1 728	3 760
Xen guest	820	750
optimized *Xen* guest	970	3 310

Table 4 shows the ultimate effect of this longer processing chain for the *Xen* VMM as well as the effect of optimizations [60]; the receiving and sending rates from a guest domain are roughly 30% and 20%, respectively, of the corresponding rates of a native Linux application. Packet multiplexing/demultiplexing accounts for about 40% and 30% of the communication overhead for the incoming traffic and for the outgoing traffic, respectively.

Probably, the lower receiving rate is due to the fact that communication between the backend interface in the driver domain and the HLVI (High Level Virtual Interface) is not optimized.

Problem 7. *In Section 5.8 we note that several operating systems including Linux, Minix, NetBSD, FreeBSD, NetWare, and OZONE can operate as paravirtualized Xen guest operating systems running on x86, x86-64, Itanium, and ARM architectures, while VMware EX Server supports full virtualization of x86 architecture. Analyze how VMware provides the functions discussed in Table 5 for Xen.*

The following information is from
http://www.vmware.com/files/pdf/software_hardware_tech_x86_virt.pdf.

The VMkernel does not run virtual machines directly. Instead, it runs a VMM that in turn is responsible for execution of the virtual machine. Each VMM is devoted to one virtual machine. To run multiple virtual machines, the VMkernel starts multiple VMM instances. Because the VMM decouples the virtual machine from the VMkernel, it is possible to run 64-bit guest operating systems on a 32-bit VMkernel (and vice versa) as long as the underlying physical CPUs have all the required features. The logic to handle the subtleties of 32-and 64-bit guest operating systems is encapsulated within the VMM, which can take advantage of a 64-bit physical CPU to run a 64-bit guest operating system efficiently, even if the underlying VMkernel runs in32-bit mode.

The VMM implements the virtual hardware on which the virtual machine runs. This hardware includes a virtual CPU, virtualI/O devices,timers, and other devices. The virtual

Table 5: Paravirtualization strategies for virtual memory management, CPU multiplexing, and I/O devices for the original *x86 Xen* implementation.

Function	Strategy
Paging	A domain may be allocated discontinuous pages. A guest OS has direct access to page tables and handles pages faults directly for efficiency; page table updates are batched for performance and validated by *Xen* for safety.
Memory	Memory is statically partitioned between domains to provide strong isolation. `XenoLinux` implements a *balloon driver* to adjust domain memory.
Protection	A guest OS runs at a lower priority level, in ring 1, while *Xen* runs in ring 0.
Exceptions	A guest OS must register with *Xen* a description table with the addresses of exception handlers previously validated; exception handlers other than the page fault handler are identical with *x86* native exception handlers.
System calls	To increase efficiency, a guest OS must install a "fast" handler to allow system calls from an application to the guest OS and avoid indirection through *Xen*.
Interrupts	A lightweight event system replaces hardware interrupts; synchronous system calls from a domain to *Xen* use *hypercalls* and notifications are delivered using the asynchronous event system.
Multiplexing	A guest OS may run multiple applications.
Time	Each guest OS has a timer interface and is aware of "real" and "virtual" time.
Network and I/O devices	Data is transferred using asynchronous I/O rings; a ring is a circular queue of descriptors allocated by a domain and accessible within *Xen*.
Disk access	Only *Dom0* has direct access to IDE and SCSI disks; all other domains access persistent storage through the Virtual Block Device (VBD) abstraction.

CPU has three features of interest:the virtual instruction set,the virtual memory management unit(MMU), and the virtual interrupt controller(PIC or APIC). The VMM can implement each of these aspects using either software techniques or hardware techniques. The combination of techniques used to virtualize the instruction set and memory determines an execution mode.

In order to run one or more virtual machines safely on a single host, ESX must isolate the virtual machines so that they cannot interfere with each other or with the VMkernel. In particular, it must prevent the virtual machines from directly executing privileged instructions that could affect the state of the physical machine as a whole. Instead, it must intercept such instructions and emulate them so their effect is applied to the virtual machines hardware, not the physical machines hardware. For example, issuing the reboot command in a virtual machine should reboot just that virtual machine, not the entire host.

The original approach to virtualizing the (32-bit)x86 instruction set is just-in-time binary translation(BT). This approach is implemented in all versions of VMwareESX, and it is the

only approach used in VMware ESX1.x and 2.x. For specificity, because this technique virtualizes the 32-bit architecture, we call it BT32. When running a virtual machines instruction stream using binary translation, the virtual machine instructions must be translated before they can be executed.More concretely, when a virtual machine is about to execute a block of code for the first time, ESX sends this code through a just-in-time binary translator, much like a Java virtual machine translates Java byte code on the fly into native instructions. The translator in the VMM does not perform a mapping from one architecture to another, but instead translates from the full unrestricted x86 instruction set to a subset that is safe to execute. In particular, the binary translator replaces privileged instructions with sequences of instructions that perform the privileged operations in the virtual machine rather than on the physical machine. This translation enforces encapsulation of the virtual machine, while preserving the x86 semantics, as seen from the perspective of the virtual machine. To keep translation overheads low, the VMM translates virtual machine instructions the first time they are about to execute, placing the resulting translated code in a translation cache. If the same virtual machine code executes again in the future, the VMM can reuse the translated code, thereby amortizing the translation costs over all future executions. To further reduce translation cost and to minimize memory usage by the translation cache, the VMM combines binary translation of kernel code running in the virtual machine with direct execution of user mode code running in the virtual machine. This is safe because user mode code cannot execute privileged instructions. A BT-based VMM must enforce a strict boundary between the part of the address space that is used by the virtual machine and the part that is used by the VMM. The VMware VMM enforces this boundary using segmentation.

About the same time they were making the transition from 32-bit to 64-bit hardware, both Intel and AMD recognized the importance of virtualization. Both companies began designing hardware that made it easier for a VMM to run virtual machines. The first hardware designs focused on the subproblem of how to virtualize the 32-and 64-bit x86 instruction set. The Intel design, called VT-x, took on a particular importance because it provided a way to virtualize 64-bit virtual machines efficiently. VT-x and AMD-V are similar in aim but different in detail. Both designs allow a VMM to do away with binary translation while still being able to fully control the execution of a virtual machine by restricting which kinds of (privileged) instructions the virtual machine can execute without intervention by the VMM.

The x86 MMU contains two main structures: a page table walker and a content-addressable memory called a *Translation Lookaside Buffer(TLB)* to accelerate address translation lookups. When an instruction accesses a virtual address, segmentation hardware converts the virtual address to a *linear address (LA)* by adding the segment base. Then the page table walker receives the LA and traverses the page table tree to produce the corresponding physical address. When the page table walk completes, the pair is inserted into the TLB to accelerate future accesses to the same address. Accordingly, the task of the VMM is not only to virtualize memory, but to virtualize virtual memory so that the guest operating system can use virtual memory. To accomplish this task, the VMM must virtualize the x86 MMU.It does so by having the VMM remap addresses a second time, below the virtual machine, from *physical address (PA)* to machine address, to confine the virtual machine to the machine memory that the VMM and VMkernel have allowed it to use.

Problem 8. *In 2012 Intel and HP announced that Itanium architecture will be discontinued. Review the architecture discussed in Section 5.10 and identify several possible reasons for this decision.*

Itanium is a processor developed jointly by HP and Intel and based on a new architecture, Explicitly Parallel Instruction Computing (EPIC), that allows the processor to execute multiple instructions in each clock cycle. EPIC implements a form of very long instruction word (VLIW) architecture in which a single instruction word contains multiple instructions.

The *Itanium* processor has 30 functional units: six general-purpose ALUs, two integer units, one shift unit, four data cache units, six multimedia units, two parallel shift units, one parallel multiply, one population count, three branch units, two 82-bit floating-point multiply-accumulate units, and two SIMD floating-point multiply-accumulate units. A 128-bit instruction word contains three instructions; the fetch mechanism can `read` up to two instruction words per clock from the L1 cache into the pipeline. Each unit can execute a particular subset of the instruction set.

The hardware supports 64-bit addressing; it has 32 64-bit general-purpose registers numbered from R0 to R31 and 96 automatically renumbered registers, R32 through R127, used by procedure calls. When a procedure is entered, the `alloc` instruction specifies the registers the procedure can access by setting the bits of a 7-bit field that controls the register usage. An illegal `read` operation from such a register out of range returns a zero value, whereas an illegal `write` operation to it is trapped as an illegal instruction.

The *Itanium* processor supports isolation of the address spaces of different processes with eight privileged *region* registers. The *Processor Abstraction Layer (PAL)* firmware allows the caller to set the values in the region register. The hardware has an *IVA register* to maintain the address of the *interruption vector table*. The entries in this table control both the interrupt delivery and the interrupt state collection. Different types of interrupts activate different interrupt handlers pointed from this table, provided that the particular interrupt is not disabled.

The hardware supports four *privilege rings*, PL0, PL1, PL2, and PL3. Privileged instructions can only be executed by the kernel running at level PL0, whereas applications run at level PL3 and can only execute non-privileged instructions; PL2 and PL4 rings are generally not used. The VMM uses *ring compression* and runs itself at PL0 and PL1 while forcing a guest OS to run at PL2. A first problem, called *privilege leaking*, is that several non-privileged instructions allow an application to determine the current privilege level (CPL); thus, a guest OS may not accept to boot or run or may itself attempt to make use of all four privilege rings.

Several reasons could have triggered the decision to terminate the production of this sophisticated processor:

- The processor has a different instruction set than IA (Intel Architecture). This means additional costs for Intel to support the development of system software such as device drivers and for the application developers.
- There is a shift of the market from PCs to tablets and smart phones thus, Intel is wise to concentrate on processors for mobile devices.
- High-performance processors such as *Itanium* have multiple functional units but do not provide explicit support for virtualization.

Problem 9. *Read [63] and analyze the results of performance comparison discussed in Section 5.11.*

The discussion in [63] is focused on user's perspective thus, the performance measures analyzed are the throughput and the response time. The general question is whether consolidation of the applications and the servers is a good strategy for cloud computing. The specific questions examined are:

- How the performance scales up with the load?
- What is the impact of a mix of applications?
- What are the implications of the load assignment on individual servers?

There is ample experimental evidence that the load placed on system resources by a single application varies significantly in time; a time series displaying CPU consumption of a single application in time clearly illustrates this fact. As we all know, this phenomenon justifies the need for CPU multiplexing among threads/processes supported by an operating system. The concept of *application and server consolidation* is an extension of the idea of creating an aggregate load consisting of several applications and aggregating a set of servers to accommodate this load; indeed, the peak resource requirements of individual applications are very unlikely to be synchronized and the aggregate load tends to lead to a better average resource utilization.

The application used in [63] is a two-tier system consisting of an Apache Web server and a MySQL database server. A client of this application starts a session as the user browses through different items in the database, requests information about individual items,and buys or sells items. Each session requires the creation of a new thread; thus, an increased load means an increased number of threads. To understand the potential discrepancies in performance among the three systems, a performance monitoring tool reports the counters that allow the estimation of: (i) the CPU time used by a binary; (ii) the number of L2-cache misses; and (iii) the number of instructions executed by a binary.

The main conclusion drawn from the experiments is that the virtualization overhead of *Xen* is considerably higher than that of *OpenVZ* and that this is due primarily to L2-cache misses. The performance degradation when the workload increases is also noticeable for *Xen*. Another important conclusion is that hosting multiple tiers of the same application on the same server is not an optimal solution.

6 Chapter 6 - Resource Management

Problem 1. *Analyze the benefits and the problems posed by the four approaches for the implementation of resource management policies: control theory, machine learning, utility-based, market-oriented.*

The four basic mechanisms for the implementation of resource management policies are:

- *Control theory.* Control theory uses the feedback to guarantee system stability and predict transient behavior [45], [51], but can be used only to predict local rather than global behavior.
- *Machine learning.* A major advantage of machine learning techniques is that they do not need a performance model of the system [81]; this technique could be applied for coordination of several autonomic system managers as discussed in [46].
- *Utility-based.* Utility-based approaches require a performance model and a mechanism to correlate user-level performance with cost as discussed in [1].
- *Market-oriented/economic mechanisms.* Such mechanisms do not require a model of the system, e.g., combinatorial auctions for bundles of resources discussed in [77].

Virtually all optimal, or near-optimal, mechanisms to address the four classes of policies do not scale up and typically target a single aspect of resource management, e.g., admission control, but ignore energy conservation; many require complex computations that cannot be done effectively in the time available to respond. The performance models are very complex, analytical solutions are intractable, and the monitoring systems used to gather state information for these models can be too intrusive and unable to provide accurate data. Many techniques are concentrated on system performance in terms of throughput and time in system, but they rarely include energy trade-offs or QoS guarantees. Some techniques are based on unrealistic assumptions; for example, capacity allocation is viewed as an optimization problem but under the assumption that servers are protected from overload.

Problem 2. *Can optimal strategies for the five classes of policies, admission control, capacity allocation, load balancing, energy optimization, and QoS guarantees be actually implemented in a cloud? The term "optimal" is used in the sense of control theory. Support your answer with solid arguments. Optimal strategies for one class may be in conflict with optimal strategies for one or more of the other classes. Identify and analyze such cases.*

Optimal strategies cannot be implemented on a cloud because they require accurate information about the state of the system and it is infeasible to acquire this information due to the scale of the system. It is also not feasible to construct the accurate models of such systems required by control theoretical and utility-based mechanisms used for policy implementation.

Optimal strategies for QoS may conflict with the ones for energy minimization; indeed, QoS may require the servers to operate outside their optimal region for energy consumption.

Problem 3. *Analyze the relationship between the scale of a system and the policies and the mechanisms for resource management. Consider also the geographic scale of the system in your arguments.*

The scale of a system plays a critical role for the implementation of the mechanisms for different resource management policies. The centralized management of a large-scale system is infeasible.

Any management scheme must rely on information gathered by a monitoring system. The volume of this information increases with the number of systems and with time elapsed between consecutive reporting, the more frequent are the reports the larger is this volume, but the more accurate is the state information available to the resource management system. The accuracy of the information provided by the monitoring system to the resource management system is also affected by the geographic scale of the system and by the frequency of reporting state information. Very frequent reporting not only adds to the system overhead, but can also lead to instability.

Though, in theory, the homogeneity of the system reduces the complexity of the system the large number of components lead to models with a humongous state space. Even an exact simulation of such a system is extremely challenging; indeed, the simulation has to include a description of each system, its state, and the interactions with each other system.

Problem 4. *What are the limitations of the control theoretic approach discussed in Section 6.2? Do the approaches discussed in Sections 6.3 and 6.4 remove or relax some of these limitations? Justify your answers.*

The applicability of control theoretic approach is limited by the scale of the system. While the basic principles of control theory are critical for the management of cloud resources, the practical implementation of the mechanisms for the five classes of policies discussed in Section 6.1 requires: (i) a model of the system and (ii) detailed information about the state of individual system components. This explains why the work reported in the literature on control theoretic approaches to cloud management is limited to toy problems and cannot be extended to realistic systems with hundred of thousands of components.

Neither the dynamic threshold, nor the cooperation between system and application managers could remove the limitation of the approach discussed here.

Problem 5. *Multiple controllers are probably necessary due to the scale of the cloud. Is it beneficial to have system and application controllers? Should the controllers be specialized for example, some to monitor performance, others to monitor power consumption? Should all the functions we want to base the resource management policies on be integrated in a single controller and one controller be assigned to a given number of servers, or to a geographic region? Justify your answers.*

The scale of a cloud requires some form of clustered organization where each cluster is managed by local systems responsible for the implementation of the five classes of resource management policies. These cluster controllers should collaborate with one another for the implementation of global policies.

A cluster manager could consist of several sub-systems, each one responsible for one the five classes of policies: admission control, capacity allocation, load balancing, energy consumption, and Quality of Service. These sub-systems should interact with each other; for example, the one responsible for admission control should reject additional workload whenever the system is in danger of failing its QoS guarantees. Similarly, load balancing and energy minimization should work in concert; when a server is lightly loaded applications running on it should be migrated to other servers and the server should be switched to a sleep state to save energy.

Each application should be built around an application manager which decides when to request additional system resources or when to release resources that are no longer needed.

Problem 6. *In a scale-free network the degree of the nodes have an exponential distribution (see Section 7.11). A scale-free network could be used as a virtual network infrastructure for cloud computing. Controllers represent a dedicated class of nodes tasked with resource management; in a scale-free network nodes with a high connectivity can be designated as controllers. Analyze the potential benefit of such a strategy.*

A scale-free network is heterogeneous, as the degrees of the nodes, have a negative exponential distribution:

$$p(k) = \frac{1}{\zeta(\gamma, k_{min})} k^{-\gamma}. \tag{21}$$

In this expression k_{min} is the smallest degree of any vertex and for the applications we discuss in this paper $k_{min} = 1$; ζ is the Hurvitz zeta function. $p(k)$ is the probability that a vertex has degree k, in other words it is connected to other k vertices.

Thus, the number of nodes with a high degree of connectivity is relatively small as we can see from Table 6.

Table 6: A power-law distribution with degree $\gamma = 2.5$; the probability, $p(k)$, and N_k, the number of vertices with degree k, for a network with a the total number of vertices $N = 10^8$.

k	$p(k)$	N_k	k	$p(k)$	N_k
1	0.745	74.5×10^6	6	0.009	0.9×10^6
2	0.131	13.1×10^6	7	0.006	0.6×10^6
3	0.049	4.9×10^6	8	0.004	0.4×10^6
4	0.023	2.3×10^6	9	0.003	0.3×10^6
5	0.013	1.3×10^6	10	0.002	0.2×10^6

An organization where these nodes are selected as controllers has several advantages:

- There is a natural way to choose controllers, c_i is a controller if $\deg(c_i) > \kappa$, with κ a relatively large number, e.g. $\kappa = 10$.
- There is a natural way to cluster the nodes around the controllers; a node q joins the cluster built around the controller c_i at the minimum distance $d(q, c_i)$. Thus, the average distance between the controller and the servers in the cluster is minimal.
- A number of studies have shown that scale-free networks have remarkable properties such as: robustness against random failures [12], favorable scaling [2, 3], resilience to congestion [32], tolerance to attacks [80], small diameter [18], and small average path length [11].

Problem 7. *Use the start-time fair queuing (SFQ) scheduling algorithm to compute the virtual startup and the virtual finish time for two threads a and b with weights $w_a = 1$ and $w_b = 5$ when the time quantum is $q = 15$ and thread b blocks at time $t = 24$ and wakes up at time $t = 60$. Plot the virtual time of the scheduler function of the real time.*

As in Problem 6, we consider two threads with the weights $w_a = 1$ and $w_b = 5$ and the time quantum is $q = 15$, and thread b blocks at time $t = 24$ and wakes up at time $t = 60$.

Initially $S_a^0 = 0$, $S_b^0 = 0$, $v_a(0) = 0$, and $v_b(0) = 0$. The scheduling decisions are made as follows:

1. t=0: we have a tie, $S_a^0 = S_b^0$ and arbitrarily thread b is chosen to run first; the virtual finish time of thread b is

$$F_b^0 = S_b^0 + q/w_b = 0 + 15/5 = 3. \tag{22}$$

2. t=3: both threads are runnable and thread b was in service, thus, $v(3) = S_b^0 = 0$; then

$$S_b^1 = \max[v(3), F_b^0] = \max(0, 3) = 3. \tag{23}$$

   But $S_a^0 < S_b^1$ thus thread a is selected to run. Its virtual finish time is

$$F_a^0 = S_a^0 + q/w_a = 0 + 15/1 = 15. \tag{24}$$

3. t=18: both threads are runnable and thread a was in service at this time thus,

$$v(18) = S_a^0 = 0 \tag{25}$$

   and

$$S_a^1 = \max[v(18), F_a^0] = \max[0, 15] = 15. \tag{26}$$

   As $S_b^1 = 3 < 12$, thread b is selected to run; the virtual finish time of thread b is now

$$F_b^1 = S_b^1 + q/w_b = 3 + 15/5 = 6. \tag{27}$$

4. t=21: both threads are runnable and thread b was in service at this time, thus,

$$v(21) = S_b^1 = 3 \tag{28}$$

   and

$$S_b^2 = \max[v(21), F_b^1] = \max[3, 6] = 6. \tag{29}$$

   As $S_b^2 < S_a^1 = 15$, thread b is selected to run again; its virtual finish time is

$$F_b^2 = S_b^2 + q/w_b = 6 + 15/5 = 9. \tag{30}$$

5. t=24: Thread b was in service at this time, thus,

$$v(24) = S_b^2 = 6 \tag{31}$$

$$S_b^3 = \max[v(24), F_b^2] = \max[6, 9] = 9. \tag{32}$$

   Thread b is suspended till $t = 60$, thus, the thread a is activated; its virtual finish time is

$$F_a^1 = S_a^1 + q/w_a = 9 + 15/1 = 24. \tag{33}$$

6. t=39: thread a was in service and it is the only runnable thread at this time, thus,

$$v(39) = S_a^1 = 15 \tag{34}$$

and

$$S_a^1 = \max[v(39), F_a^1] = \max[15, 24] = 24. \tag{35}$$

Then,

$$F_a^2 = S_a^2 + q/w_a = 24 + 15/1 = 39. \tag{36}$$

7. t=54: thread a was in service and it is the only runnable thread at this time thus,

$$v(54) = S_a^1 = 24 \tag{37}$$

and

$$S_a^2 = \max[v(54), F_a^2] = \max[24, 39] = 39. \tag{38}$$

Then,

$$F_a^2 = S_a^3 + q/w_a = 39 + 15/1 = 54. \tag{39}$$

8. t=69: thread a was in service at this time, thus,

$$v(69) = S_a^2 = 39 \tag{40}$$

and

$$S_a^4 = \max[v(69), F_a^2] = \max[39, 54] = 54. \tag{41}$$

But now thread b is runnable and $S_b^3 = 9$.

Thus, thread b is activated and

$$F_b^3 = S_b^3 + q/w_b = 9 + 15/5 = 12. \tag{42}$$

Problem 8. *Apply the borrowed virtual time (BVT) scheduling algorithm to the problem in Example 2 of Section 6.11, but with a time warp of* $W_c = -30$.

There are two threads a and b of best effort applications. The first thread has a weight twice of the weight of the second, $w_a = 2w_b$; when $k_a = 180$ and $k_b = 90$, then $\Delta = 90$.

We consider periods of real time allocation of $C = 9$ *mcu*; the two threads a and b are allowed to run for $2C/3 = 6$ *mcu* and $C/3 = 3$ *mcu*, respectively.

Threads a and b are activated at times

$$\begin{array}{ll} a: & 0, 5, 5+9 = 14, 14+9 = 23, 23+9 = 32, 32+9 = 41, \ldots \\ b: & 2, 2+9 = 11, 11+9 = 20, 20+9 = 29, 29+9 = 38, \ldots \end{array} \tag{43}$$

The context switches occur at real times

$$2, 5, 11, 14, 20, 23, 29, 32, 38, 41, \ldots \tag{44}$$

A real-time thread c with a time warp $W_c = -30$ wakes up at time $t = 9$ and wakes up periodically at times $t = 18, 27, 36, \ldots$ for 3 units of time and is competing with the two best-effort threads a and b.

Table 7: The real time and the effective virtual time of the three threads of each context switch are shown.

Context switch	Real time	Running thread	Effective virtual time of the running thread
1	t=2	a	$E_a(2) = A_a(2) = A_a(0) + \Delta/3 = 0 + 90/3 = 30$
2	t=5	b	$E_b^1 = A_b^1 = A_b^0 + \Delta = 0 + 90 = 90 \;\Rightarrow\; a$ *runs next*
3	t=9	a	c wakes up $E_a^1 = A_a^1 + 2\Delta/3 = 30 + (-60) = 90$ $[E_a(9), E_b(9), E_c(9)] = (90, 90, -30) \;\Rightarrow\; c$ *runs next*
4	t=12	c	$SVT(12) = \min(90, 90)$ $E_c^s(12) = SVT(12) + W_c = 90 + (-30) = 60$ $E_c(12) = E_c^s(12) + \Delta/3 = 60 + 30 = 90 \;\Rightarrow\; b$ *runs next*
5	t=14	b	$E_b^2 = A_b^2 = A_b^1 + 2\Delta/3 = 90 + 60 = 150 \;\Rightarrow\; a$ *runs next*
6	t=18	a	c wakes up $E_a^3 = A_a^3 = A_a^2 + 2\Delta/3 = 90 + 60 = 150$ $[E_a(18), E_b(18), E_c(18)] = (150, 150, 60) \;\Rightarrow\; c$ *runs next*
7	t=21	c	$SVT = \min(150, 150)$ $E_c^s(21) = SVT + W_c = 150 + (-30) = 120$ $E_c(21) = E_c^s(21) + \Delta/3 = 90 + 30 = 120 \;\Rightarrow\; b$ *runs next*
8	t=23	b	$E_b^3 = A_b^3 = A_b^2 + 2\Delta/3 = 150 + 60 = 210 \;\Rightarrow\; a$ *runs next*
9	t=27	a	c wakes up $E_a^4 = A_a^4 = A_a^3 + 2\Delta/3 = 150 + 60 = 210$ $[E_a(27), E_b(27), E_c(27)] = (210, 210, 120) \;\Rightarrow\; c$ *runs next*
10	t=30	c	$SVT = \min(210, 210)$ $E_c^s(30) = SVT + W_c = 210 + (-30) = 180$ $E_c(30) = E_c^s(30) + \Delta/3 = 180 + 30 = 210 \;\Rightarrow\; b$ *runs next*
11	t=32	b	$E_b^4 = A_b^4 = A_b^3 + 2\Delta/3 = 210 + 60 = 270 \;\Rightarrow\; a$ *runs next*
10	t=36	a	c wakes up $E_a^5 = A_a^5 = A_a^4 + 2\Delta/3 = 210 + 60 = 270$ $[E_a(36), E_b(36), E_c(36)] = (270, 270, 210) \;\Rightarrow\; c$ *runs next*
12	t=39	c	$SVT = \min(270, 270)$ $E_c^s(39) = SVT + W_c = 270 + (-30) = 240$ $E_c(39) = E_c^s(39) + \Delta/3 = 240 + 30 = 270 \;\Rightarrow\; b$ *runs next*
13	t=41	b	$E_b^5 = A_b^5 = A_b^4 + 2\Delta/3 = 270 + 60 = 330 \;\Rightarrow\; a$ *runs next*

Table 7 summarizes the evolution of the system when the real-time application thread c competes with the two best-effort threads a and b. Context switches occur now at real times

$$t = 2, 5, 9, 12, 14, 18, 21, 23, 27, 30, 32, 36, 39, 41, \ldots \tag{45}$$

The context switches at times

$$t = 9, 18, 27, 36, \ldots \tag{46}$$

are triggered by the waking up of the thread c which preempts the currently running thread. At $t = 9$ the time warp $W_c = -60$ gives priority to thread c. Indeed

$$E_c(9) = A_c(9) + W_c = 0 + (-30) = -30 \tag{47}$$

compared with $E_a(9) = 90$ and $E_b(9) = 90$. The same conditions occur every time the real-time thread wakes-up. The best-effort application threads have the same effective virtual time when the real-time application thread finishes and the scheduler chooses b to be dispatched first.

7 Chapter 7 - Networking

Problem 1. *Four ground rules for an open-architecture principles are cited in the "Brief history of the Internet." Read the paper and analyze the implication of each one of these rules.*

The paper [54] co-authored by B.M. Leiner, V. G. Cerf, D. D. Clark, R. E. Khan, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts and S. Wolff who can be called "the fathers of the Internet" lists the following ground rules:

1. Each distinct network would have to stand on its own and no internal changes could be required to any such network to connect it to the Internet.
2. Communications would be on a best effort basis. If a packet didn't make it to the final destination, it would shortly be retransmitted from the source.
3. Black boxes would be used to connect the networks; these would later be called gateways and routers. There would be no information retained by the gateways about the individual flows of packets passing through them, thereby keeping them simple and avoiding complicated adaptation and recovery from various failure modes.
4. There would be no global control at the operations level.

(1) This ground rule led the design of a *network of networks* an open system where networks with different architecture, physical implementations, properties, could operate under a minimum set of assumptions. The only thing they had to have in common are the interpretation of an IP address and the network protocol. The hourglass architecture in Figure 11 illustrates this concept.

(2) The switching technology available in late 60s, the fact that routers would connect links with different speeds and had to operate asynchronously, that different networks would have different characteristics, the "best effort" model was the only feasible model for the Internet. Error detection and retransmission was feasible as the primary mechanism for error control because, initially, the Arpanet (the original name of what is now called the Internet) was a "data network," a network designed to carry only data as opposed to its today's use to transport data with real-time constraints such as video and audio streaming.

(3) Basing the network architecture on packet switching rather than circuit switching was a critical design decision and showed the forward thinking of this group of people. They understood at that time that digital technologies for processing, storing and communication will evolve at a very fast pace and they will allow a deeper integration of computing and communication.

(4) No large-scale system can be based on centralized control and this was another great insight; they could not have even dreamed of the number of hosts and networks in the Internet today.

Problem 2. *The paper [54] lists also several key issues for the design of the network:*

1. *Algorithms to prevent lost packets from permanently disabling communications and enabling them to be successfully retransmitted from the source.*

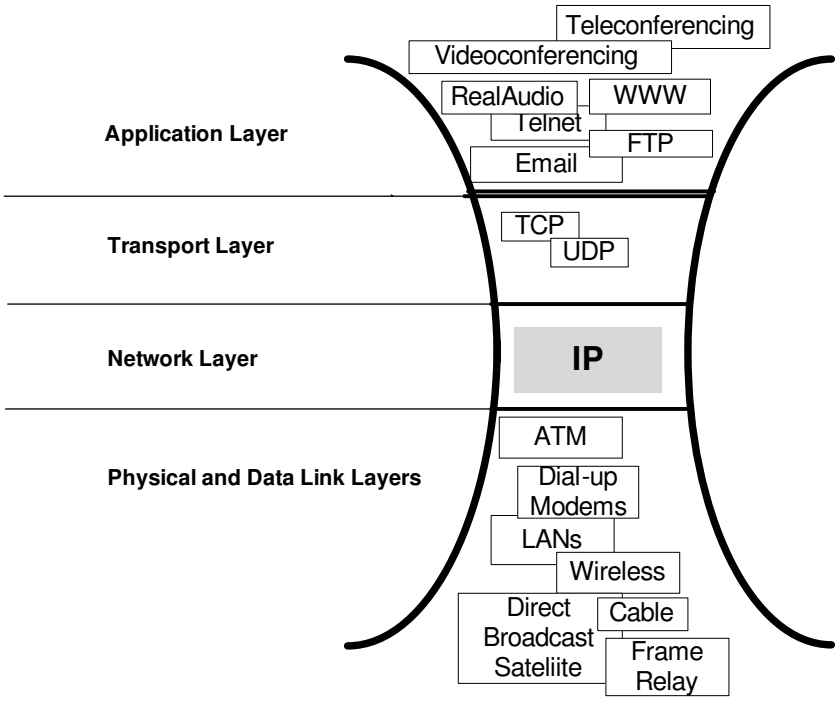

Figure 11: The hourglass network architecture of the Internet. Regardless of the application, the transport protocol, and the physical network, all packets are routed through the Internet from the source to the destination by the IP protocol; routing is based on the destination IP address.

2. *Providing for host-to-host "pipelining" so that multiple packets could be enroute from source to destination at the discretion of the participating hosts, if the intermediate networks allowed it.*

3. *The need for end-end checksums, reassembly of packets from fragments and detection of duplicates, if any.*

4. *The need for global addressing.*

5. *Techniques for host-to-host flow control.*

Discuss how these issues were addressed by the TCP/IP network architecture.

(1) Error control: the detection and the eventual correction of errors including packets in error, lost packets, and duplicate packets, is critical for the development of any communication infrastructure. The principle formulated by (1) encouraged the development of a simple protocol stack (as opposed to the protocol stack suggested by the OSI reference model) where error control is supported at the data link, the transport, and at the application layer.

(2) The Internet protocol stack allows two hosts to communicate using a variety of physical communication channel, data link, transport, and application protocols. Two hosts can communicate through any number of such "virtual channels," each one using a different element

from each family of protocols. Moreover two applications running on a host can communicate concurrently as each send and receives messages at its own port, see Figure 12.

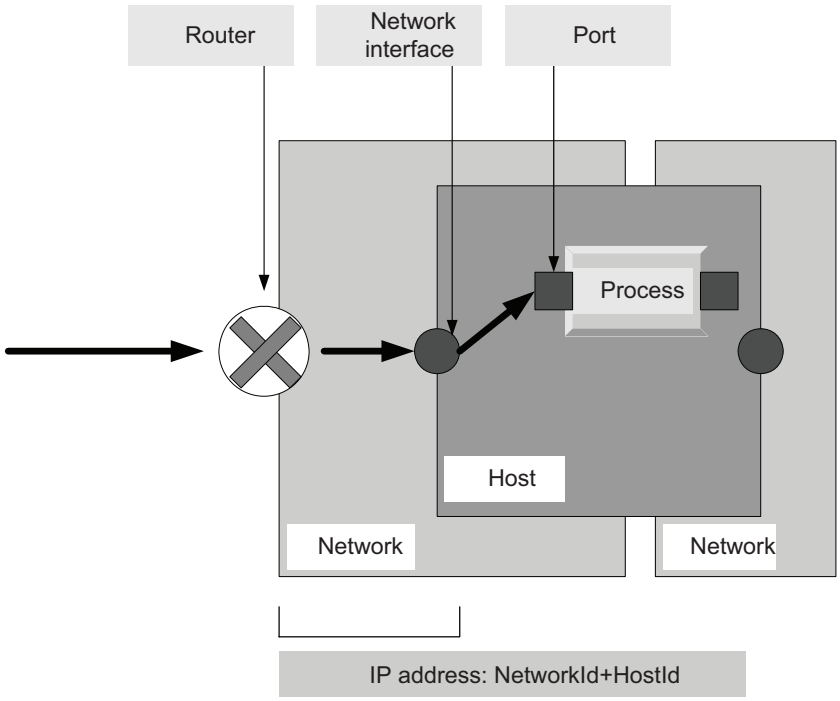

Figure 12: Packet delivery to processes and threads; the packet is first routed by the IP protocol to the destination network and then to the host specified by the IP address. Each application listens to an abstraction of the end point of a logical communication channel called a *port*.

(3) The Internet is a network of networks, each one with its own internal organization and accepts packets of different size. Thus, the need for fragmentation and reassembly. The routers in each network have differen hardware capabilities and may drop packets, thus, the need for error control. The different links speed and routing delay led to the possibility of duplicate packets, as senders timeout and retransmit packets.

(4) The need for global addressing is a basic requirement in a network of networks, as the Internet. Without global addressing it would be impossible for hosts in different networks to communicate. The Internet uses a hierarchical addressing scheme; an IP address consists of a concatenation of the network address and the host address in that network.

Problem 3. *Analyze the challenges of transition to IPv6. What will be in your view the effect of this transition on cloud computing?*

The Internet Protocol, Version 4 (IPv4), provides an addressing capability of 2^{32}, or approximately 4.3 billion addresses, a number that proved to be insufficient. Indeed, the Internet Assigned Numbers Authority (IANA) assigned the last batch of 5 address blocks to the Regional Internet Registries in February 2011, officially depleting the global pool of completely fresh blocks of addresses; each of the address blocks represents approximately 16.7 million possible addresses.

The migration to IPv6 involves upgrading all applications, hosts, routers, and DNS infrastructure; also, moving to IPv6 requires backward compatibility, any organization migrating to IPv6 should maintain a complete IPv4 infrastructure.

A group from Google has investigated in 2009 the adoption of IPv6 [19] and has concluded that: "...the IPv6 deployment is small but growing steadily, and that adoption is still heavily influenced by a small number of large deployments. While we see IPv6 adoption in research and education networks, IPv6 deployment is, with one notable exception, largely absent from consumer access networks."

Problem 4. *Discuss the algorithms used to compute the TCP window size.*

TCP uses a sliding window flow control protocol; if W is the window size, then the data rate S of the sender is:

$$S = \frac{W \times MSS}{RTT} \text{ bps} = \frac{W}{RTT} \text{ packets/second}$$

where MSS and RTT denote the Maximum Segment Size and the Round Trip Time, respectively. If S is too small the transmission rate is smaller than the channel capacity, while a large S leads to congestion. The channel capacity in the case of communication over the Internet is not a fixed quantity but, as different physical channels are shared among many flows, it depends on the load of the network.

The actual window size W is affected by two factors: (a) the ability of the receiver to accept new data and (b) the sender's estimation of the available network capacity. The receiver specifies the amount of additional data it is willing to accept in the *receive window* field of every frame; the receive window shifts when the receiver receives and acknowledges a new segment of data. When a receiver advertises a window size of zero, the sender stops sending data and starts the persist timer; this timer is used to avoid the deadlock when a subsequent window size update from the receiver is lost. When the persist timer expires, the sender sends a small packet and the receiver responds by sending another acknowledgement containing the new window size. In addition to the flow control provided by the receiver, the sender attempts to infer the available network capacity and to avoid overloading the network. The source uses the losses and the delay to determine the level of congestion. If $awnd$ denotes the receiver window and $cwnd$ the congestion window set by the sender, the actual window should be:

$$W = \min(cwnd, awnd).$$

Several algorithms are used to calculate $cwnd$ including Tahoe and Reno, developed by Jacobson in 1988 and 1990. Tahoe was based on slow start (SS), congestion avoidance (CA), and fast retransmit (FR); the sender probes the network for spare capacity and detects congestion based on loss. The slow start means that the sender starts with a window of two times MSS, $init\_cwnd = 1$; for every packet acknowledged, the congestion window increases by 1 MSS so that the congestion window effectively doubles for every RTT. When the congestion window exceeds the threshold, $cwnd \geq ssthresh$, the algorithm enters the congestion avoidance state; in CA state, on each successful acknowledgment $cwnd \leftarrow cwnd + 1/cwnd$ and on each RTT $cwnd \leftarrow cwnd + 1$. The fast retransmit is motivated by the fact that the timeout is too long thus, a sender retransmits immediately after 3 duplicate acknowledgments without waiting for a timeout. Two adjustments are then made:

$$flightsize = \min(awnd, cwnd) \quad \text{and} \quad ssthresh \leftarrow \max(flightsize/2, 2)$$

and the system enters in the slow start state, $cwnd = 1$.

The pseudocode describing the Tahoe algorithm is:

```
for every ACK {
        if (W < ssthresh) then W++      (SS)
        else    W += 1/W                (CA)
    }
    for every loss {
        ssthresh = W/2
            W  = 1
    }
```

Problem 5. *Creating a virtual machine (VM) reduces ultimately to copying a file, therefore the explosion of the number of VMs cannot be prevented, see Section 9.7. As each VM needs its own IP address, virtualization could drastically lead to an exhaustion of the IPv4 address space. Analyze the solution to this potential problem adopted by the IaaS cloud service delivery model.*

A client must know the IP address of a virtual machine in the cloud, to be able to connect to it. Domain Name Service (DNS) is used to map human-friendly names of computer systems to IP addresses in the Internet or in private networks. DNS is a hierarchical distributed database and plays a role reminiscent of a phone books in the Internet. In late 2010 Amazon announced a DNS service called *Route 53* to route users to AWS services and to infrastructure outside of AWS. A network of DNS servers scattered across the globe, enables customers to gain reliable access to AWS and place strict controls over who can manage their DNS system by allowing integration with AWS Identity and Access Management (IAM).

For several reasons, including security and the ability of the infrastructure to scale up, the IP addresses of instances visible to the outside world are mapped internally to private IP addresses. A virtual machine running under Amazon's *EC2* has several IP addresses:

1. *EC2 Private IP Address:* The internal address of an instance; it is only used for routing within the *EC2* cloud.

2. *EC2 Public IP Address:* Network traffic originating outside the AWS network must use either the public IP address or the elastic IP address of the instance. The public IP address is translated using the Network Address Translation (NAT) to the private IP address when an instance is launched and it is valid until the instance is terminated. Traffic to the public address is forwarded to the private IP address of the instance.

3. *EC2 Elastic IP Address:* The IP address allocated to an AWS account and used by traffic originated outside AWS. NAT is used to map an elastic IP address to the private IP address. Elastic IP addresses allow the cloud user to mask instance or availability zone failures by programmatically re-mapping a public IP addresses to any instance associated with the user's account. This allows fast recovery after a system failure; for example, rather than waiting for a cloud maintenance team to reconfigure or replace the failing host, or waiting for DNS to propagate the new public IP to all of the customers of

a web service hosted by EC2, the web service provider can re-map the elastic IP address to a replacement instance. Amazon charges a fee for unallocated Elastic IP addresses.

Problem 6. *Read the paper describing the stochastic fair queuing algorithm [25]. Analyze the similarities and dissimilarities of this algorithm and the start-time fair queuing discussed in Section 6.10.*

The class-based queuing (CBQ) algorithm was proposed by Sally Floyd and Van Jacobson in 1995 [25]. The objective of CBQ is to support flexible link sharing for applications that require bandwidth guarantees such as VoIP, video streaming, and audio streaming. At the same time, CBQ supports some balance between short-lived network flows, such as Web searches, and long-lived ones, such as video streaming or file transfers.

CBQ aggregates the connections and constructs a hierarchy of classes with different priorities and throughput allocations. To accomplish link sharing, CBQ uses several functional units: (i) a *classifier* that uses the information in the packet header to assign arriving packets to classes; (ii) an *estimator* of the short-term bandwidth for the class; (iii) a *selector*, or scheduler, which identifies the highest-priority class to send next and, if multiple classes have the same priority, to schedule them on a round-robin basis; and (iv) a *delayer* to compute the next time when a class that has exceeded its link allocation is allowed to send.

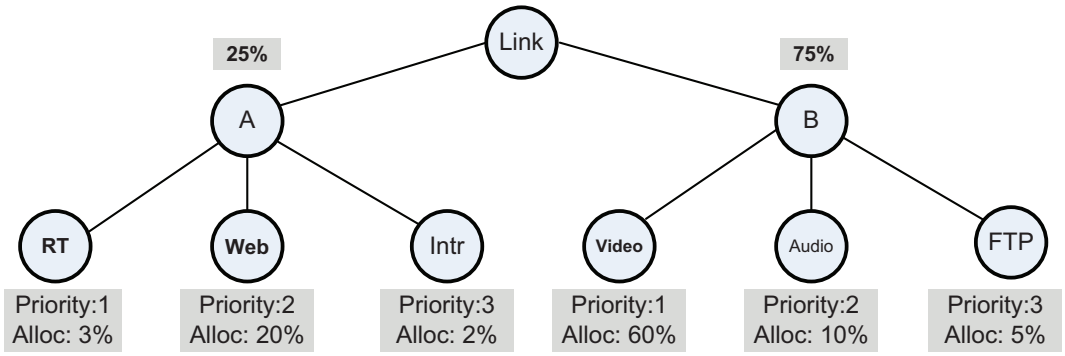

Figure 13: CBQ link sharing for two groups A, of short-lived traffic, and B, of long-lived traffic, allocated 25% and 75% of the link capacity, respectively. There are three classes with priorities 1, 2, and 3; the RT (real-time) and the video streaming have priority one and are allocated 3% and 60%, respectively, web transactions and audio streaming have priority 2 and are allocated 20% and 10%, respectively, and Intr (interactive applications) and FTP (file transfer protocols) have priority 3 and are allocated 2% and 5%, respectively, of the link capacity.

The classes are organized in a tree-like hierarchy; for example, in Figure 13 we see two types of traffic: group A corresponding to short-lived traffic and group B corresponding to long-lived traffic. The leaves of the tree are considered Level 1 and in this example they include six classes of traffic: real time, Web, interactive, video streaming, audio streaming, and file transfer. At Level 2 there are the two classes of traffic, A and B. The root, at Level 3, is the link itself.

The link-sharing policy aims to ensure that if sufficient demand exists, then, after some time intervals, each interior or leaf class receives its allocated bandwidth. The distribution

of the "excess" bandwidth follows a set of guidelines, but does not support mechanisms for congestion avoidance.

A class is *overlimit* if over a certain recent period it has used more than its bandwidth allocation (in bytes per second), *underlimit* if it has used less, and *atlimit* if it has used exactly its allocation. A leaf class is *satisfied* if it is underlimit and has a persistent backlog, and it is *unsatisfied* otherwise. A nonleaf class is unsatisfied if it is underlimit and has some descendent class with a persistent backlog. A precise definition of the term *persistent backlog* is part of a local policy. A class does not need to be *regulated* if it is underlimit or if there are no unsatisfied classes. The class should be regulated if it is overlimit and if some other class is unsatisfied, and this regulation should continue until the class is no longer overlimit or until there are no unsatisfied classes.

Similarities: Both protocols support: (i) a mix of workload classes; (ii) workloads with real-time constraints.

Dissimilarities: CBQ: (i) organizes the workload classes in a tree structure; (ii) allows borrowing among classes.

Problem 7. *The small-worlds networks were introduced by D. Watts and S. H. Strogatz. They have two desirable features, high clustering and small path length. Read [85] and design an algorithm to construct a small-worlds network.*

The basic idea: starting from a ring lattice with n vertices and m edges per node, rewire each edge at random with probability $0 \leq p \leq 1$; when $p = 0$ the graph is regular, and when $p = 1$ the graph is random.

We start with a ring connecting the n nodes and chose at random $m < n/2$ nodes as the starting points of rewiring, call this set $\mathcal{R}_s$; then chose at random $m < n/2$ nodes as the end points, call this set $\mathcal{R}_e$, such that $\mathcal{R}_s \cap \mathcal{R}_e = \emptyset$. Then, the probability of rewiring is $p = 2m/n$. Pick up at random a node $a_i \in \mathcal{R}_s$ and a node $b_i \in \mathcal{R}_e$ and construct the link (a_i, b_i). Then remove the two nodes from the sets: $\mathcal{R}_s = \mathcal{R}_s - a_i$ and $\mathcal{R}_e = \mathcal{R}_e - b_i$ and continue the process until $\mid \mathcal{R}_s \mid = \mid \mathcal{R}_3 \mid = 0$.

Problem 8. *The properties of scale-free networks are discussed in [2, 3, 4, 11]. Discuss the important features of systems interconnected by scale-free networks discussed in these papers.*

A number of studies have shown that scale-free networks have remarkable properties such as: robustness against random failures [12], favorable scaling [2, 3], resilience to congestion [32], tolerance to attacks [80], small diameter [18], and small average path length [11].

Problem 9. *Discuss the three models for the propagation of an infectious disease in a finite population, SI, SIR, and SIS. Justify the formulas describing the dynamics of the system for each model.*

Several classes of epidemic algorithms exist; the concepts used to classify these algorithms *Susceptible, Infective* and *Recovered* refer to the state of the individual in the population subject to infectious disease and, by extension, to the recipient of information in a distributed system:

- Susceptible - the individual is healthy but can get infected; the system does not know the specific information, but can get it.

- Infective - the individual is infected and able to infect others; the system knows the specific information and uses the rules to disseminate it.
- Recovered - the individual is infected but does not infect others; the system knows the specific information and does not disseminate it.

Susceptible-Infective (SI). The SI algorithms apply when the entire population is susceptible to be infected; once an individual becomes infected it remains in that state until the entire population is infected. If $I(t)$ is the number of individuals infected, $S(t)$ is the number of individuals susceptible to be infected, and $R(t)$ the number of those infected and then recovered at time t and if all individuals have an equal probability β of contracting the disease then,

$$\frac{I(0)}{2[1-I(0)]} \leq I(t) \leq \frac{I(0)}{1-I(0)} e^{-\beta t} \tag{48}$$

and

$$\frac{1}{2}\left(\frac{1}{S(0)-1}\right) e^{-\beta t} \leq S(t) \leq \left(\frac{1}{S(0)-1}\right) e^{-\beta t} \tag{49}$$

when we assume that $I(t)$ and $S(t)$ are continuous variables, rather than natural numbers.

Susceptible-Infectious-Recover (SIR). SIR algorithms are based on the model developed by Kermack and McKendrik in 1927 [48]. The model assumes the following transition from one state to another $S \mapsto I \mapsto R$; it also assumes that $1/\gamma$ is the average infectious period and that the size of the population is fixed

$$S(t) + I(t) + R(t) = N. \tag{50}$$

The dynamics of the model is captured by the following equations

$$\frac{dS(t)}{dt} = -\beta S(t) I(t), \quad \frac{dI(t)}{dt} = \beta S(t) I(t) - \gamma I(t), \quad \text{and} \quad \frac{dR(t)}{dt} = \gamma I(t). \tag{51}$$

Susceptible-Infective-Susceptible (SIS). SIS algorithms [13] are particular cases of SIR models when individuals recover from the disease without immunity. If $p = R(r)/I(r)$, then the number of newly infected grows until $(1-p)/2$ are infected and then decreases exponentially to $(1-p)$ according to the expression

$$I(r) = \frac{1-p}{1 + \left(\frac{(1-p)N}{I(0)} - 1\right)} \times N. \tag{52}$$

Recall that the m-th moment of the power-law distribution of a discrete random variable X, $P_X(x=k) = k^{-\gamma}$ is

$$E\left[X^m\right] = \sum_{k=1}^{\infty} k^m P_X(x=k) = \sum_{k=1}^{\infty} k^m k^{-\gamma} = \sum_{k=1}^{\infty} \frac{1}{k^{\gamma-m}}. \tag{53}$$

For power-law networks, the epidemic threshold λ for the *Susceptible–Infectious–Recovered* (SIR) and *Susceptible–Infectious–Susceptible* (SIS) epidemic models can be expressed as [22]

$$\lambda = \frac{E[X]}{E[X^2]}. \tag{54}$$

The *epidemic threshold* is defined as the minimum ratio of infected nodes to the cured nodes per time such that it still allows the epidemics to continue without outside infections. It follows that $\lambda \mapsto 0$ if $\gamma \in (2, 3)$; in other words, such networks become infinitely susceptible to epidemic algorithms. This property is very important for dissemination of control information in such networks.

8 Chapter 8 - Storage

Problem 1. *Analyze the reasons for the introduction of storage area networks (SANs) and their properties. Hint: read [61].*

According to [61]: "IT organizations were attempting to *regain control* over the dispersed assets characteristic of client/server computing. The data center took on a renewed importance in most enterprises. To that end, multiple servers in a machine room sought the capability to access their backend storage without necessarily having individual storage directly attached and therefore dedicated to an individual server. This caused the emergence of storage area networks (SANs). SANs had the advantage that storage systems could be separately engineered from the servers, allowing the pooling of storage (statically configured) and resulting in improved efficiencies and lower risks for the customer (storage investment was not tied to a particular server hardware or software choice). SAN technology opened up new opportunities in simplified connectivity, scalability, and cost and capacity manageability.

Fibre Channel became the predominant networking technology and large storage systems, such as IBM Total Storage Enterprise Storage Server (ESS), support this protocol and use it to attach multiple servers. ...The widespread adoption and commoditization of Ethernet LANs running TCP/IP (Transport Control Protocol/Internet Protocol) has caused increasing interest in the use of this technology in the SAN. The unifying of networking under TCP/IP and Ethernet offers the benefits of standardization, increased rate of innovation, and lower costs, in both hardware and device support. Management costs are reduced since staffs need to be familiar with only one type of network,rather than two."

Problem 2. *Block virtualization simplifies the storage management tasks in SANs. Provide solid arguments in support of this statement. Hint: read [61].*

According to [61]: "Although the availability of networked storage provides improved access to data, it still leaves some key issues unaddressed. SANs enable arbitrary connectivity between using system and storage, but they still pose operational problems. Although a SAN may not be hard-wired to its attached storage, it is still *softwire* in the sense that the configuration is typically static, and changes cannot be made to the attached storage or using system without disruption. The addition of a layer of indirection between the using system and storage, provided either through a hardware switch or an intermediary software-based system, allows the using system (typically a server) to deal with non-disruptive changes in the storage configuration.

Additionally, virtual volumes can be created that cut across multiple storage devices, this capability is referred to as *block virtualization.* Block virtualization allows all the storage on a SAN to be pooled and then allocated for access by the using systems. This also simplifies various storage management tasks, such as copy services, varying storage devices on-line or off-line, and so on."

Problem 3. *Analyze the advantages of memory-based checkpointing. Hint: read [43].*

Most of the clusters checkpoint the snapshots to the stable storage of the cluster. A stable storage generally means non-volatile disk storage, where the I/O bottleneck is increasingly becoming an identified performance bottleneck of HPC even without the burden of checkpointing/restarting. In addition, checkpointing snapshots are usually large, and the cost of storing/retrieving them to/from disks is high. With large systems composed of hundreds of

thousands of nodes, the conventional disk-based checkpointing will overwhelm the slow storage devices. Memory-based checkpointing is a promising solution to break through the bottleneck from the stable storage. However, due to its complexity and potential threat to reliability, memory-based checkpointing is rarely supported by the mainstream of current checkpointing systems.

There are two main diskless checkpointing schemes: parity-based and neighbor-based checkpointing. The parity-based approach checkpoints the applications image to each nodes spare memory. The parity of all the checkpoints is calculated and saved in extra checkpointing processors to make the parallel application recoverable in the event of failures. The neighbor-based approach removes the coding/decoding calculation of the parity-based approach by introducing one more copy of checkpointing image, i.e. each node checkpoints the application image to both its own memory and that of the neighbors.

Problem 4. *Discuss the security of distributed file systems including SUN NFS, Apollo Domain, Andrew, IBM AIX, RFS, and Sprite. Hint: read [69].*

From [69]:

"NFS. NFS uses the underlying Unix file protection mechanism on servers for access checks. Each RPC request from a client conveys the identity of the user on whose behalf the request is being made. The server temporarily assumes this identity, and file accesses that occur while servicing the request are checked exactly as if the user had logged in directly to the server. The standard Unix protection mechanism using user, group and world mode bits is used to specify protection policies on individual files and directories. In the early versions of NFS, mutual trust was assumed between all participating machines. The identity of a user was determined by a client machine and accepted without further validation by a server. The level of security of an NFS site was effectively that of the least secure system in the environment. To reduce vulnerability, requests made on behalf of the root (the Unix superuser) on a workstation were treated by the server as if they had come from a non-existent user, nobody. Root thus received the lowest level of privileges for remote files. More recent versions of NFS can be configured to provide a higher level of security. DES-based mutual authentication is used to validate the client and the server on each RPC request. Since file data in RPC packets is not encrypted, NFS is still vulnerable to unauthorized release and modification of information if the network is not physically secure. The common DES key needed for mutual authentication is obtained from information stored in a publicly readable database. Stored in this database for each user and server is a pair of keys suitable for public key encryption. One key of the pair is stored in the clear, while the other is stored encrypted with the login password of the user. Any two entities registered in the database can deduce a unique DES key for mutual authentication.

Apollo Domain. Security in DOMAIN is predicated on the physical integrity of Apollo workstations and on the trustworthiness of the kernels on them. Since the network is also assumed to be secure, communications on it are sent in the clear. The network component of the kernel at each node uses a special field in the header of every packet to indicate whether the originator of the packet was a user-level program, or the kernel itself. This prevents user-level programs from masquerading as trusted system software. A distributed user registry stores each users login password in encrypted form, as in Unix. When a user logs in on a node, the local kernel encrypts the password typed by the user, fetches his login entry from the registry, and validates the user by comparing the two encrypted passwords. Each instance of a logged-in user is associated with a unique identifier, called a PPON, that identifies the user, the project, and the organization he belongs to, and the node at which this login instance

occurred. The PPON is used on all access checks on behalf of this logged-in instance of the user. Nodes cache recently-used registry information to enhance availability. The user registry, called RGY, is a replicated database with one master site and multiple read-only slaves for availability. Each replica contains the entries for all users in the system. Updates are made at the master site which then propagates them asynchronously to the slave sites. Direct authentication to the master, using a Needham-Schroeder authentication handshake, is required before an update can be performed. Protection policies are specified by access lists on objects. An access list entry maps a PPON to a set of rights. Components of PPONs can be wildcarded. If multiple entries are applicable in a given access check, the most specific matching entry overrides the others. Checking of access has been done at the home node of objects in some releases of DOMAIN, and at the usage node in other releases. These are logically equivalent, since the kernels trust each other.

Andrew. The design of Andrew pays serious attention to security, while ensuring that the mechanisms for enforcing it do not inhibit legitimate use of the system. Security is predicated on the integrity of a small number of Vice servers. These servers are physically secure, are accessible only to trusted operators, and run only trusted system software. Neither the network, nor workstations are trusted by Vice. Authentication and secure transmission mechanisms based on end-to-end encryption are used to provide secure access to servers from workstations. It is still the responsibility of a user to ensure that he is not being compromised by malicious software on his workstation. To protect himself against Trojan horse attacks, a concerned user has to maintain the physical integrity of his workstation and to deny remote access to it via the network. The protection domain in Andrew is composed of users, corresponding to human users, and groups, which are sets of users and other groups. Membership in a group is inherited, and a user accumulates the privileges of all the groups he belongs to directly and indirectly. Inheritance of membership simplifies the maintenance and administration of the protection domain. Membership in a special group called System:Administrators endows administrative privileges, including unrestricted access to any file in the system. Andrew uses an access list mechanism for protection. The total rights specified for a user are the union of all the rights collectively specified for him and for all the groups of which he is a direct or indirect member. An access list can specify negative rights. An entry in a negative rights list indicates denial of the specified rights, with denial overriding possession in case of conflict. Negative rights are intended primarily as means of rapidly and selectively revoking access to critical files and directories. For conceptual simplicity, Vice associates access lists with directories rather than files. The access list applies to all files in the directory, thus giving them uniform protection status. In addition, the three owner bits of the Unix file mode are used to indicate readability, writability or executability. In Andrew, these bits indicate what can be done to the file rather than who can do it."

Problem 5. *The designers of the Google file system (GFS) have re-examined the traditional choices for a file system. Discuss observations regarding these choices that have guided the design of GFS. Hint: read [31].*

The Google File System (GFS) was developed in the late 1990s. It uses thousands of storage systems built from inexpensive commodity components to provide petabytes of storage to a large user community with diverse needs [31]. It is not surprising that a main concern of the GFS designers was to ensure the reliability of a system exposed to hardware failures, system software errors, application errors, and last but not least, human errors. The system was designed after a careful analysis of the file characteristics and of the access models. Some

of the most important aspects of this analysis reflected in the GFS design are:

- Scalability and reliability are critical features of the system; they must be considered from the beginning rather than at some stage of the design.
- The vast majority of files range in size from a few GB to hundreds of TB.
- The most common operation is to `append` to an existing file; random `write` operations to a file are extremely infrequent.
- Sequential `read` operations are the norm.
- The users process the data in bulk and are less concerned with the response time.
- The consistency model should be relaxed to simplify the system implementation, but without placing an additional burden on the application developers.

Several design decisions were made as a result of this analysis:

1. Segment a file in large chunks.
2. Implement an atomic file `append` operation allowing multiple applications operating concurrently to `append` to the same file.
3. Build the cluster around a high-bandwidth rather than low-latency interconnection network. Separate the flow of control from the data flow; schedule the high-bandwidth data flow by pipelining the data transfer over TCP connections to reduce the response time. Exploit network topology by sending data to the closest node in the network.
4. Eliminate caching at the client site. Caching increases the overhead for maintaining consistency among cached copies at multiple client sites and it is not likely to improve performance.
5. Ensure consistency by channeling critical file operations through a *master*, a component of the cluster that controls the entire system.
6. Minimize the involvement of the *master* in file access operations to avoid hot-spot contention and to ensure scalability.
7. Support efficient checkpointing and fast recovery mechanisms.
8. Support an efficient garbage-collection mechanism.

Problem 6. *In his seminal paper on the virtues and limitations of the transaction concept [35] Jim Gray analyzes logging and locking. Discuss the main conclusions of his analysis.*

Jim Gray writes in [35]: "Logging and locking are an alternative implementation of the transaction concept. The legendary Greeks, Ariadne and Theseus, invented logging. Ariadne gave Theseus a magic ball of string which he unraveled as he searched the Labyrinth for the Minotaur. Having slain the Minotaur, Theseus followed the string back to the entrance rather then remaining lost in the Labyrinth. This string was his log allowing him to undo the process of entering the Labyrinth. But the Minotaur was not a protected object so its death

was not undone by Theseus exit. Hansel and Gretel copied Theseus trick as they wandered into the woods in search of berries. They left behind a trail of crumbs that would allow them to retrace their steps by following the trail backwards, and would allow their parents to find them by following the trail forwards. This was the first undo and redo log. Unfortunately, a bird ate the crumbs and caused the first log failure.

The basic idea of logging is that every undoable action must not only do the action but must also leave behind a string, crumb or undo log record which allows the operation to be undone. Similarly, every redoable action must not only do the operation but must also generate a redo log record which allows the operations to be redone. Based on Hansel and Gretels experience, these log records should be made out of strong stuff (not something a bird would eat). In computer terms, the records should be kept in stable storage usually implemented by keeping the records on several non-volatile devices, each with independent failure modes. Occasionally, a stable copy of each object should be recorded so that the current state may be reconstructed from the old state.

The log records for database operations are very simple. They have the form:

```
NAME OF TRANSACTION:
PREVIOUS LOG RECORD OF THIS TRANSACTION:
NEXT LOG RECORD OF THIS TRANSACATION:
TIME:
TYPE OF OPERATION:
OBJECT OF OPERATION:
OLD VALUE:
NEW VALUE:
```

The old and new values can be complete copies of the object, but more typically they just encode the changed parts of the object. For example, an update of a field of a record of a file generally records the names of the file, record and field along with the old and new field values rather than logging the old and new values of the entire file or entire record. The log records of a transaction are threaded together. In order to undo a transaction, one undoes each action in its log. This technique may be used both for transaction abort issued by the program and for cleaning up after incomplete (uncommitted) transactions in case of a system problem such as deadlock or hardware failure. In the event that the current state of an object is lost, one may reconstruct the current state from an old state in stable storage by using the redo log to redo all recent committed actions on the old state. Some actions need not generate log records. Actions on unprotected objects (e.g. writing on a scratch file), and actions which do not change the object state (e.g. reads of the object) need not generate log records.

On the other hand, some actions must initially only generate log records which will be applied at transaction commit. A real action which cannot be undone must be deferred until transaction commit. In a log-based system, such actions are deferred by keeping a redo log of deferred operations. When the transaction successfully commits, the recovery system uses this log to do the deferred actions for the first time. These actions are named (for example by sequence number) so that duplicates are discarded and hence the actions are restartable (see below). Another detail is that the undo and redo operations must be restartable, that is, if the operation is already undone or redone, the operation should not damage or change the object state. The need for restartability comes from the need to deal with failures during undo and redo processing. Restartability is usually accomplished with version numbers (for disc pages) and with sequence numbers (for virtual circuits or sessions). Essentially, the undo

or redo operation reads the version or sequence number and does nothing if it is the desired number. Otherwise, it transforms the object and the sequence number.

In a log-based scheme, transaction commit is signaled by writing the commit record to the log. If the transaction has contributed to multiple logs then one must be careful to assure that the commit appears either in all logs or in none of the logs. Multiple logs frequently arise in distributed systems since there are generally one or more logs per node. They also arise in central systems. The simplest strategy to make commit an atomic action is to allow only the active node of the transaction to decide to commit or abort (all other participants are slaves and look to the active node for the commit or abort decision).

It is generally desirable to allow each participant in a transaction to unilaterally abort the transaction prior to the commit. If this happens, all other participants must also abort. The *two-phase commit* protocol is intended to minimize the time during which a node is not allowed to unilaterally abort a transaction. It is very similar to the wedding ceremony in which the minister asks "Do you?" and the participants say "I do" (or "No way!") and then the minister says "I now pronounce you", or "The deal is off". At commit, the two-phase commit protocol gets agreement from each participant that the transaction is prepared to commit. The participant abdicates the right to unilaterally abort once it says "I do" to the prepare request. If all agree to commit, then the commit coordinator broadcasts the commit message. If unanimous consent is not achieved, the transaction aborts. Many variations on this protocol are known (and probably many more will be published). If transactions run concurrently, one transaction might read the outputs (updates or messages) of another transaction. If the first transaction aborts, then undoing it requires undoing the updates or messages read by the second transaction. This in turn requires undoing the second transaction. But the second transaction may have already committed and so cannot be undone. To prevent this dilemma, real and protected updates (undoable updates) of a transaction must be hidden from other transactions until the transaction commits. To assure that reading two related records, or rereading the same record, will give consistent results, one must also stabilize records which a transaction reads and keep them constant until the transaction commits. Otherwise a transaction could reread a record and get two different answers."

Problem 7. *Michael Stonebreaker argues that "blinding performance depends on removing overhead..." Discuss his arguments regarding the NoSQL concept. Hint: read [78].*

Michael Stonebreaker argues in [78]: "There are two possible reasons to move to either of these alternate DBMS technologies: performance and flexibility. The performance argument goes something like the following. I started with MySQL for my data storage needs and over time found performance to be inadequate. My options were:

- *Shard*[3] my data to partition it across several sites, giving me a serious headache managing distributed data in my application or
- Abandon MySQL and pay big licensing fees for an enterprise SQL DBMS or move to something other than a SQL DBMS.

The flexibility argument goes something like the following. My data does not conform to a rigid relational schema. Hence, I cant be bound by the structure of a RDBMS and need something more flexible.

[3]A database shard is a horizontal partition in a database or search engine. Each individual partition is referred to as a shard or database shard.

For simplicity, we will focus this discussion on the workloads for which NoSQL databases are most often considered: update- and lookup-intensive OLTP workloads, not query-intensive data warehousing workloads. We do not consider document repositories or other specialized workloads for which NoSQL systems may be well suited. There are two ways to improve OLTP performance; namely, provide automatic *sharding* over a shared-nothing processing environment and improve per-server OLTP performance. In the first case, one improves performance by providing scalability as nodes are added to a computing environment; in the second case, one improves performance of individual nodes. Every serious SQL DBMS (e.g., Greenplum, Asterdata, Vertica, Paraccel, etc.) written in the last 10 years has provided shared nothing scalability, and any new effort would be remiss if it did not do likewise. Hence, this component of performance should be *table stakes* for any DBMS. In my opinion, nobody should ever run a DBMS that does not provide automatic sharding over computing nodes. As a result, this posting continues with the other component, namely, single node OLTP performance.

The overhead associated with OLTP databases in traditional SQL systems has little to do with SQL, which is why NoSQL is such a misnomer. Instead, the major overhead in an OLTP SQL DBMS is communicating with the DBMS using ODBC or JDBC. Essentially all applications that are performance sensitive use a stored-procedure interface to run application logic inside the DBMS and avoid the crippling overhead of back-and-forth communication between the application and the DBMS. The other alternative is to run the DBMS in the same address space as the application, thereby giving up any pretense of access control or security. Such embeddable DBMSs are reasonable in some environments, but not for mainstream OLTP, where security is a big deal. Using either stored procedures or embedding, the useful work component is a very small percentage of total transaction cost, for todays OLTP data bases which usually fit in main memory. Instead, a recent paper calculated that total OLTP time was divided almost equally between the following four overhead components:

- Logging: Traditional databases write everything twice; once to the database and once to the log. Moreover, the log must be forced to disk, to guarantee transaction durability. Logging is, therefore, an expensive operation.
- Locking: Before touching a record, a transaction must set a lock on it in the lock table. This is an overhead-intensive operation.
- Latching: Updates to shared data structures (B-trees, the lock table, resource tables, etc.) must be done carefully in a multi-threaded environment. Typically, this is done with short-term duration latches, which are another considerable source of overhead.
- Buffer Management: Data in traditional systems is stored on fixed-size disk pages. A buffer pool manages which set of disk pages is cached in memory at any given time. Moreover, records must be located on pages and the field boundaries identified. “

Problem 8. *Discuss the Megastore data model. Hint: read [9].*

Megastore is scalable storage for online services. The system, distributed over several data centers, has a very large capacity, 1 PB in 2011, and it is highly available. The basic design philosophy of the system is to partition the data into *entity groups* and replicate each partition independently in data centers located in different geographic areas. The entity groups are

application-specific and store together logically related data. For example, an email account could be an entity group for an email application. Data should be carefully partitioned to avoid excessive communication between entity groups. Sometimes it is desirable to form multiple entity groups, as in the case of blogs [9]. The system supports full ACID semantics within each partition and provides limited consistency guarantees across partitions.

Another distinctive feature of the system is the use of the Paxos consensus algorithm to replicate primary user data, metadata, and system configuration information across data centers and for locking. The version of the Paxos algorithm used by *Megastore* does not require a single master. Instead, any node can initiate `read` and `write` operations to a write-ahead log replicated to a group of symmetric peers.

The middle ground between traditional and *NoSQL* databases taken by the *Megastore* designers is also reflected in the data model. The data model is declared in a *schema* consisting of a set of *tables* composed of *entries*, each entry being a collection of named and typed *properties*. The unique primary key of an entity in a table is created as a composition of entry properties. A *Megastore* table can be a *root* or a *child* table. Each *child entity* must reference a special entity, called a *root entity* in its root table. An entity group consists of the primary entity and all entities that reference it.

Problem 9. *Discuss the use of locking in the Bigtable. Hint: read [17] and [14].*

BigTable is a distributed storage system developed by Google to store massive amounts of data and to scale up to thousands of storage servers [17]]. The system uses the Google File System to store user data as well as system information. To guarantee atomic `read` and `write` operations, it uses the *Chubby* distributed lock service; the directories and the files in the namespace of *Chubby* are used as locks.

The system is based on a simple and flexible data model. It allows an application developer to exercise control over the data format and layout and reveals data locality information to the application clients. Any `read` or `write` row operation is atomic, even when it affects more than one column. The column keys identify *column families*, which are units of access control. The data in a column family is of the same type. Client applications written in C++ can add or delete values, search for a subset of data, and look up data in a row.

9 Chapter 9 - Security

Problem 1. *Identify the main security threats for the SaaS cloud delivery model on a public cloud. Discuss the different aspects of these threats on a public cloud vis-a-vis the threats posed to similar services provided by a traditional service-oriented architecture running on a private infrastructure.*

The favorite means of attack are: distributed denial of service (DDDS) attacks which prevent legitimate users to access cloud services, phishing[4], SQL injection[5], or cross-site scripting[6].

Availability of cloud services is another major concern. System failures, power outages, and other catastrophic events could shutdown cloud services for extended periods of time. *Insecure APIs* may not protect the users during a range of activities starting with authentication.

Problem 2. *Analyze how the six attack surfaces discussed in Section 9.1 and illustrated in Figure 14 apply to the SaaS, PaaS and IaaS cloud delivery models.*

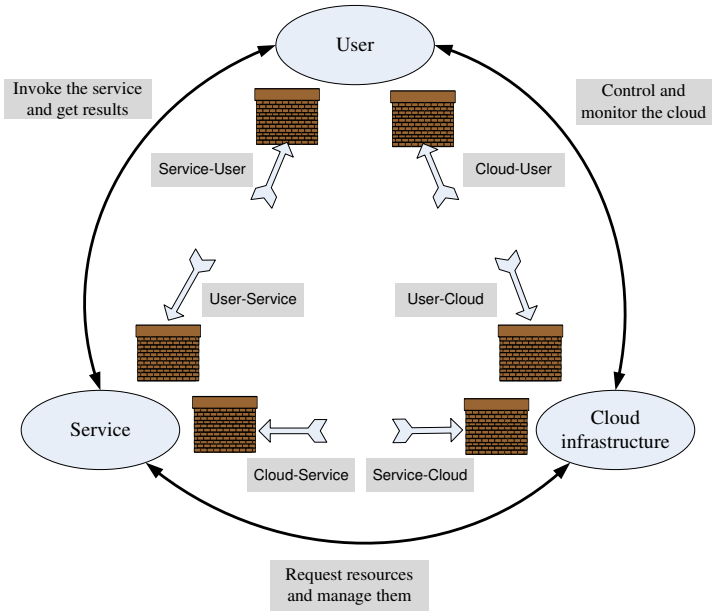

Figure 14: Surfaces of attacks in a cloud computing environment.

An attempt to identify and classify the attacks in a cloud computing environment is discussed in [38]. The three actors involved in the model considered are: the user, the

[4]Phishing is an attack aiming to gain information from a site database by masquerading as a trustworthy entity; such information could be names and credit card numbers, social security numbers, other personal information stored by online merchants or other service providers.

[5]SQL injection is a form of attack typically used against a web site; then an SQL command entered in a web form causes the contents of a database used by the web site to be dumped to the attacker or altered. SQL injection can be used against other transaction processing systems and it is successful when the user input is not strongly typed and/or rigourously filtered.

[6]Cross-site scripting is the most popular form of attack against web sites; a browser permits the attacker to insert client-scripts into the web pages and thus, bypass the access controls at the web site.

service, and the cloud infrastructure, and there are six types of attacks possible, see Figure 14. The user can be attacked from two directions, the service and the cloud. SSL certificate spoofing, attacks on browser caches, or phishing attacks are example of attacks that originate at the service. The user can also be a victim of attacks that either originate at the cloud or spoofs to originate from the cloud infrastructure. In the case of *SaaS* attacks from the cloud infrastructure and from the service are less likely.

The service can be attached from the user, this is rather common for the *SaaS*. Buffer overflow, SQL injection, and privilege escalation are the common types of attacks for the service. The service can also be subject to attacks by the cloud infrastructure and this is probably the most serious line of attack; this is less likely in the case of *SaaS*. Limiting access to resources, privilege-related attacks, data distortion, injecting additional operations, are only a few of the many possible lines of attacks originated at the cloud.

The cloud infrastructure can be attacked by a user which targets the cloud control system; this type of attack is common for all cloud delivery models. The types of attacks are the same a user directs toward any other cloud service. The cloud infrastructure may also be targeted by a service requesting an excessive amount of resources and causing the exhaustion of the resources; this is not realistic for the *SaaS* cloud delivery model.

Problem 3. *Analyze Amazon privacy policies and design a service level agreement you would sign on if you were to process confidential data using AWS.*

The main aspects of privacy are: the lack of user control, potential unauthorized secondary use, data proliferation, and dynamic provisioning [65]. The lack of user control refers to the fact that user-centric data control is incompatible with cloud usage. Once data is stored on the servers of the CSP, the user losses control on the exact location and, in some instances, it could lose access to the data.

The types of information Amazon gathers:

1. Information given by the users themselves: "We receive and store any information you enter on our Web site or give us in any other way. You can choose not to provide certain information, but then you might not be able to take advantage of many of our features. We use the information that you provide for such purposes as responding to your requests, customizing future shopping for you, improving our stores, and communicating with you."

2. Automatic Information: "We receive and store certain types of information whenever you interact with us. For example, like many Web sites, we use "cookies," and we obtain certain types of information when your Web browser accesses Amazon.com or advertisements and other content served by or on behalf of Amazon.com on other Web sites."

3. Mobile:" When you download or use apps created by Amazon or our subsidiaries, we may receive information about your location and your mobile device, including a unique identifier for your device. We may use this information to provide you with location-based services, such as advertising, search results, and other personalized content. Most mobile devices allow you to turn off location services."

4. E-mail Communications: "To help us make e-mails more useful and interesting, we often receive a confirmation when you open e-mail from Amazon.com if your computer

supports such capabilities. We also compare our customer list to lists received from other companies, in an effort to avoid sending unnecessary messages to our customers. If you do not want to receive e-mail or other mail from us, please adjust your Customer Communication Preferences. Information from Other Sources: We might receive information about you from other sources and add it to our account information."

A user can:

- Adjust the *Customer Communication Preferences* to prevent Emails from Amazon.
- Adjust the *Advertising Preferences* to prevent Amazon from using personal information that gathered to allow third parties to personalize advertisements Amazon display to the user.
- Disable cookies using the facilities provided by the browsers.

Problem 4. *Analyze the implications of the lack of trusted paths in commodity operating systems and give one or more examples showing the effects of this deficiency. Analyze the implications of the two-level security model of commodity operating systems.*

Specialized *closed-box platforms* such as the ones on some cellular phones, game consoles, and ATM (Automatic Teller Machines) could have embedded cryptographic keys that allow themselves to reveal their true identity to remote systems and authenticate the software running on them. Such facilities are not available to an *open-box platforms*, the traditional hardware designed for commodity operating systems.

The two-level security model supports two modes of operation, a kernel and a user mode. The kernel mode is a privileged mode, it allows a user unrestricted access to all system resources and the ability to perform any operation it wishes to perform. This explains why malicious individuals try to hijack a system and operate in kernel mode, then use the system to attack other systems in the Internet. The two-level security model creates serious problems for virtualization when there are three distinct modes operation modes: hypervisor, kernel of a guest operating system, and user. This model does not support the requirements of a system when a group of individuals have different privilege levels as is the case in a corporate environments.

Problem 5. *Compare the benefits and the potential problems due to virtualization on public, private, and hybrid clouds.*

There is little doubt that the public and hybrid clouds are more exposed to security risks as the user population is more diverse and it is easy for a malicious user to get access to the systems.

Virtualization is critical for the *IaaS* paradigm:

1. It supports multi-tenency thus, improved utilization of the servers.
2. It allows users to replicate on the cloud the environment they are familiar with, rather than forcing them to work in idiosyncratic environments.

3. It enables workload migration thus, helps increase the system ability to react to component failures and simplifies system management, supports application scaling, and reduces costs.

4. In principle it could improve security as a hypervisor - a less complex software than a traditional operating system - provides an additional layer of control and can prevent an application to exploit weaknesses of an operating system.

In a 2005 paper [28] Garfinkel and Rosenblum argue that the serious implications of virtualization on system security cannot be ignored. This theme is revisited in 2008 by Price [66] who reaches similar conclusions.

A first type of undesirable effects of virtualization lead to a diminished ability of an organization to manage its systems and track their status:

- The number of physical systems in the inventory of an organization is limited by cost, space, energy consumption, and human support. Creating a virtual machine (VM) reduces ultimately to copying a file, therefore the explosion of the number of VMs is a fact of life. The only limitation for the number of VMs is the amount of storage space available.

- In addition to quantity there is also a qualitative aspect of the explosion of the number of VMs. Traditionally, organizations install and maintain the same version of system software. In a virtual environment such a homogeneity cannot be enforced; thus, the number of different operating systems, their versions, and the patch status of each version will be very diverse and this heterogeneity will tax the support team.

- Probably one of the most critical problems posed by virtualization is related to the software lifecycle. The traditional assumption is that the software lifecycle is a straight line, hence the patch management is based on a monotonic forward progress. The virtual execution model *maps to a tree structure* rather than a line; indeed, at any point in time multiple instances of the VM can be created and then each one of them can be updated, different patches installed, and so on. This problem has serious implication on security as we shall see shortly.

Direct implications of virtualization on security - a first question is how can the support team deal with the consequences of an attack in a virtual environment. Do we expect the infection with a computer virus or a worm to be less manageable in a virtual environment? The surprising answer to this question is that an infection may last indefinitely. Some of the infected VMs may be dormant at the time when the measures to clean up the systems are taken and then, at a later time, wake up and infect other systems; the scenario can repeat itself and guarantee that infection will last indefinitely. This is in stark contrast with the manner an infection is treated in non-virtual environments; once an infection is detected, the infected systems are quarantined and then cleaned-up; the systems will then behave normally until the next episode of infection occurs.

The more general observation is that in a traditional computing environment a steady state can be reached. In this steady state all systems are brought up to a "desirable" state, while "undesirable" states, states when some of the systems are either infected by a virus or display an undesirable pattern of behavior, are only transient. This desirable state is reached by installing the latest version of the system software

and then applying to all systems the latest patches. Due to the lack of control, a virtual environment may never reach such a steady state. In a non-virtual environment the security can be compromised when an infected laptop is connected to the network protected by a firewall, or when a virus is brought in on a removable media. But, unlike a virtual environment, the system can still reach a steady state.

A side effect of the ability to record in a file the complete state of a VM is the possibility to roll back a VM. This opens wide the door for a new type of vulnerability caused by events recorded in the memory of an attacker. Two such situations are discussed in [28]. The first situation is that one-time passwords are transmitted in clear and the protection is guaranteed only if the attacker does not have the possibility to access passwords used in previous sessions. If a system runs the S/KEY password system[7] then an attacker can replay rolled back versions and access past sniffed passwords. The second situation is related to the requirement of some cryptographic protocols and even non-cryptographic protocols regarding the "freshness" of the random number source used for session keys and nonces. This situation occurs when a VM is rolled back to a state when a random number has been generated but not yet used. Even non-cryptographic use of random numbers may be affected by the rollback scenario. For example, the initial sequence number for a new TCP connection must be "fresh;" when it is not, then the door to TCP hijacking is left open.

Another undesirable effect of virtual environment affects the trust. Each computer system in a network has a unique physical, or MAC address; the uniqueness of this address guarantees that an infected or a malicious system can be identified and then cleaned, shut down, or denied network access. This process breaks down for virtual systems when VMs are created dynamically; often, to avoid name collision, a random MAC address is assigned to a new VM. Another effect is that popular VM images are shared by many users.

The ability to guarantee confidentiality of sensitive data is yet another pilar of security affected by virtualization. Virtualization undermines the basic principle that the time sensitive data is stored on any system should be reduced to a minimum. First, the owner has very limited control on where sensitive data is stored, it could be spread across many servers, and may be left on some of them indefinitely. To be able to roll it back a VMM records the state of a VM; this process allows an attacker to access sensitive data the owner attempted to destroy.

Problem 6. *Read [10] and discuss the measures taken by Amazon to address the problems posed by shared images available from AWS. Would it be useful to have a cloud service to analyze images and signed them before being listed and made available to the general public?*

A user of AWS has the option to choose between Amazon Machine Images (AMIs) accessible through the *Quick Start* or the *Community AMI* menus of the *EC2* service. The option of using one of these AMIs is especially tempting for a first time user, or for a less sophisticated one.

A recent paper reports on the results of an analysis carried over a period of several months, from November 2010 to May 2011 of over five thousand AMIs available through the public catalog at Amazon [10]. Many of the images analyzed allowed a user to *undelete* files, recover credentials, private keys, or other types of sensitive information with little effort and using

[7] S/KEY is a password system based on Leslie Lamport scheme. It is used by several operating systems including, Linux, OpenBSD, and NetBSD. The real password of the user is combined with a short set of characters and a counter that is decremented at each use to form a single-use password.

standard tools. The results of this study were shared with the Amazon's Security Team which acted promptly to reduce the threats posed to AWS users.

The details of the testing methodology can be found in [10], here we only discuss the results. The study was able to audit some $5,303$ images out of the $8,448$ Linux AMIs and $1,202$ Windows AMIs at Amazon sites in the US, Europe and Asia. The audit covered software vulnerabilities and security and privacy risks.

The average duration of an audit was 77 minutes for a Windows image and 21 minutes for a Linux image; the average disk space used was about 1 GB and 2.7 GB, respectively. The entire file system of a Windows AMI was audited because most of the malware targets Windows systems. Only directories containing executables for Linux AMIs were scanned; this strategy and the considerably longer start-up time of Windows explain the time discrepancy of the audits for the types of AMIs.

The *software vulnerability* audit revealed that 98% of the Windows AMIs (249 out of 253) and 58% (2005 out of $3,432$) Linux AMIs audited had critical vulnerabilities. The average number of vulnerabilities per AMI were 46 for Windows and 11 for Linux AMIs. Some of the images were rather old; $145, 38$, and 2 Windows AMIs and $1197, 364$, and 106 Linux AMIs were older than $2, 3$, and 4 years, respectively. The tool used to detect vulnerabilities, *Nessus*, available from *http://www.tenable.com/productus/nessus*, classifies the vulnerabilities based on their severity in four groups, at levels $0 - 3$. The audit reported only vulnerabilities of the highest severity level, e.g., remote code execution.

Three types of *security risks* are analyzed: (1) backdoors and leftover credentials, (2) unsolicited connections, and (3) malware. An astounding finding is that about 22% of the scanned Linux AMIs contained credentials allowing an intruder to remotely login to the system. Some 100 passwords, 995 `ssh` keys, and 90 cases when both could be retrieved were identified.

To rent a Linux AMI a user must provide the public part of the her `ssh` key and this key is stored in the `authorized_keys` in the home directory. This opens a backdoor for a malicious creator of an AMI who does not remove her own public key from the image; then this malicious user and can remotely login to any instance of this AMI. Another backdoor is opened when the `ssh` server allows password-based authentication and the malicious creator of an AMI who does not remove her own password; this backdoor is even wider open as one can extract the password hashes and then crack the passwords using a tool such as John the Riper (see *http://www.openwall.com/john*).

Another threat is posed by the omission of the `cloud-init` script that should be invoked when the image is booted. This script provided by the Amazon regenerates the host key an `ssh` server uses to identify itself; the public part of this key is used to authenticate the server. When this key is shared among several systems, these systems become vulnerable to *man-in-the middle*[8] attacks. When this script does not run, an attacker can use the *NMap* tool[9] to match the `ssh` keys discovered in the AMI images with the keys obtained with *NMap*.

[8] In a *man-in-the middle* an attacker impersonates the agents at both ends of a communication channel and makes them believe that they communicate through a secure channel. For example, if B sends her public key to A, but C is able to intercept it, such an attack proceeds as follows: C sends a forged message to A claiming to be from B, but instead includes C's public key. Then A encrypts her message with C's key, believing that she is using B's key, and sends the encrypted message to B. The intruder, C, intercepts, deciphers the message using her private key, possibly alters the message, and re-encrypts with the public key B originally sent to A. When B receives the newly encrypted message, she believes it came from A.

[9] *NMap* is a security tool running on most operating systems including *Linux, Microsoft Windows, Solaris,*

The study reports that the authors were able to identify more than 2,100 instances following this procedure.

Unsolicited connections pose a serious threat to a system. Outgoing connections allow an outside entity to receive privileged information, e.g., the IP address of an instance and events recorded by a *syslog* daemon to files in the *var/log* directory of a Linux system. Such information is available only to users with administrative privileges. The audit detected two Linux instances with modified *syslog* daemon which forwarded to an outside agent information about events such as login and incoming requests to a web server. Some of the unsolicited connections are legitimate, for example, connections to a software update site. It is next to impossible to distinguish legitimate from malicious connections.

Malware including viruses, worms, spyware, and trojans were identified using *ClamAV*, a software tool with a database of some 850,000 malware signatures, available from *http://www.clamav.net*. Two infected Windows AMIs were discovered, one with a *Trojan-Spy* (variant 50112) and a second one with a *Trojan-Agent* (variant 173287). The first trojan carries out key logging, and allows stealing data from the files system and monitoring processes; the AMI also included a tool to decrypt and recover passwords stored by the *Firefox* browser, called *Trojan.Firepass*.

The creator of a shared AMI assumes some *privacy risks*; her private keys, IP addresses, browser history, shell history, and deleted files can be recovered from the published images. A malicious agent can recover the AWS API keys which are not password protected. Then the malicious agent can start AMIs and run cloud applications at no cost to herself, as the computing charges are passed on to the owner of the API key. The search can target files with names such as *pk-[0-9A-Z]*.pem* or *cert-[0-9A-Z]*.pem* used to store API keys.

Another avenue for a malicious agent is to recover `ssh` keys stored in files named *id_dsa* and *id_rsa*. Though `ssh` keys can be protected by a *passphrase*[10] the audit determined that the majority of *ssh* keys (54 out of 56) were not password protected.

Recovery of IP addresses of other systems owned by the same user requires access to the *lastlog* or the *lastb* databases. The audit found 187 AMIs with a total of more than 66,000 entries in their *lastb* databases. Nine AMIs contained Firefox browser history and allowed the auditor identify the domains contacted by the user.

612 AMIs contained at least one shell history file. The audit analyzed 869 history files named *∼/.history*, *∼/.bash_history*, and *∼/.sh_history* containing some 160,000 lines of command history and identified 74 identification credentials. The users should be aware that, when the *HTTP* protocol is used to transfer information from a user to a web site, the *GET* requests are stored in the logs of the web server. Passwords and credit card numbers communicated via a *GET* request can be exploited by a malicious agent with access to such logs. When remote credentials, such as the DNS management password, are available then a malicious agent can redirect traffic from its original destination to her own system.

Recovery of deleted files containing sensitive information pose another risk for the provider of an image. When the sectors on the disk containing sensitive information are actually overwritten by another file, recovery of sensitive information is much harder. To be safe, the

HP-UX, SGI-IRIX and BSD variants such as *Mac OS X* to map the network. Mapping the network means to discover hosts and services in a network.

[10] A *passphrase* is a sequence of words used to control access to a computer system; it is the analog of a password, but provides added security. For high security non-military applications NIST recommends an 80-bit strength passphrase. Hence a secure passphrase should consist of at least 58 characters including uppercase and alphanumeric characters. The entropy of the written English is less than 1.1 bits per character.

creator of the image effort should use utilities such as `shred, scrub, zerofree`, or `wipe` to make recovery of sensitive information next to impossible. If the image is created with the block-level tool discussed at the beginning of this section, the image will contain blocks of the file system marked as free; such block may contain information from deleted files. The audit process was able to recover files from 98% of the AMIs using the `exundelete` utility. The number of files recovered from an AMI were as low as 6 and as high as $40,000$.

Problem 7. *Analyze the risks posed by foreign mapping and the solution adopted by Xoar. What is the security risk posed by XenStore?*

In the original implementation of *Xen* a service running in a *DomU* sends data to, or receives data from a client located outside the cloud using a network interface in *Dom0*; it transfers the data to I/O devices using a device driver in *Dom0*[11]. Therefore, we have to ensure that run time communication through *Dom0* is encrypted. Yet, Transport Layer Security (TLS) does not guarantee that *Dom0* cannot extract cryptographic keys from the memory of the OS and applications running in *DomU*. A significant security weakness of *Dom0* is that the entire state of the system is maintained by *XenStore*. A malicious VM can deny access to this critical element of the system to other VMs; it can also gain access to the memory of a *DomU*. This brings us to additional requirements for confidentiality and integrity imposed on *Dom0*.

Dom0 should be prohibited to use foreign mapping for sharing memory with a *DomU*, unless *DomU* initiates the procedure in response to a hypercall from *Dom0*. When this happens, *Dom0* should be provided with an encrypted copy of the memory pages and of the virtual CPU registers. The entire process should be closely monitored by the hypervisor which, after the access, should check the integrity of the affected *DomU*.

Xoar is a modified version on *Xen* designed to boost system security[20]. The security model of *Xoar* assumes that the system is professionally managed and that a privileged access to the system is granted only to system administrators. The model also assumes that the administrators have neither financial incentives, nor the desire to violate the trust of the user. The security threats come from a guest VM which could attempt to violate the data integrity or the confidentiality of another guest VM on the same platform, or to exploit the code of the guest. Another source of threats are bugs in initialization code of the management virtual machine.

Users of *Xoar* are able to only share service VMs with guest VMs that they control; to do so they specify a tag on all of the devices of their hosted VMs. Auditing is more secure, whenever a VM is created, deleted, stopped, or restarted by *Xoar* the action is recorded in an append-only database on a different server accessible via a secure channel.

Problem 8. *Read [20] and discuss the performance of the system. What obstacles to its adoption by the providers of IaaS services can you foresee?*

The authors of [20] write: "The performance of *XOAR* is evaluated against a stock *Xen Dom0* in terms of memory overhead, I/O throughput, and overall system performance. Each service VM in Xoar runs with a single virtual CPU; in stock Xen Dom0 runs with 2 virtual CPUs, the configuration used in the commercial XenServer platform. All figures are the average of three runs, with 95% confidence intervals shown where appropriate. Our test system was a Dell Precision T3500 server, with a quad-core 2.67 GHz Intel Xeon W3520

[11] Later implementations of *Xen* offer the pass through option.

processor, 4 GB of RAM, a Tigon 3 Gigabit Ethernet card, and an Intel 82801JIR SATA controller with a Western Digital WD3200AAKS-75L9A0 320 GB 7200 RPM disk. VMX, EPT, and IOMMU virtualization are enabled. We use Xen 4.1.0 and Linux 2.6.314 PVOPS kernels for the tests. Identical guests running an Ubuntu 10.04 system, configured with two VCPUs, 1 GB of RAM and a 15 GB virtual disk are used on both systems. For network tests, the system is connected directly to another system with an Intel 82567LF-2 Gigabit network controller.

- Memory overhead. Figure 15 shows the memory requirements of each of the components in *Xoar*. Systems with multiple network or disk controllers can have several instances of NetBack and BlkBack. Also, since users can select the service VMs to run, there is no single figure for total memory consumption. In commercial hosting solutions, console access is largely absent rendering the Console redundant. Similarly, PCIBack can be destroyed after boot. As a result, the memory requirements range from 512 MB to 896 MB, assuming a single network and block controller, representing a saving of 30% to an overhead of 20% on the default 750 MB Dom0 configuration used by XenServer. All performance tests compare a complete configuration of Xoar with a standard *Dom0 Xen* configuration.

Component	Memory	Component	Memory
XenStore-Logic	32 MB	XenStore-State	32 MB
Console	128 MB	PCIBack	256 MB
NetBack	128 MB	BlkBack	128 MB
Builder	64 MB	Toolstack	128 MB

Figure 15: Memory requirements of individual XOAR components.

- I/O performance. Disk performance is tested using Postmark, with VMs virtual disks backed by files on a local disk. Figure 16 shows the results of these tests with different configuration parameters.

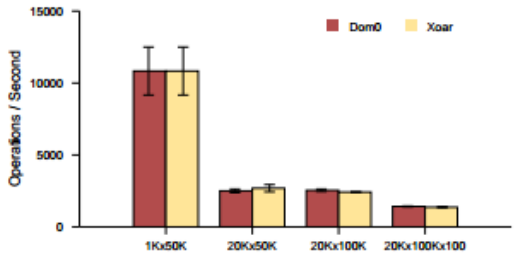

Figure 16: Disk performance using Postmark (higher is better). The x-axis denotes (files × transactions × subdirectories)..

Network performance is tested by fetching a 512 MB and a 2 GB file across a gigabit LAN using `wget`, and writing it either to disk, or to /dev/null (to eliminate performance artifacts due to disk performance). Figure 17 shows these results.

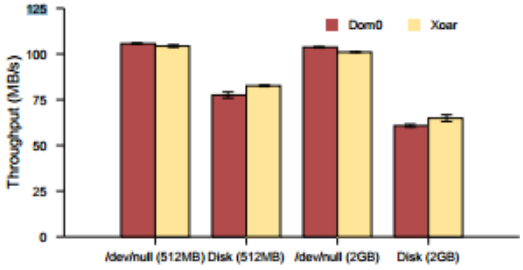

Figure 17: Network performance with `wget` (higher is better).

Overall, disk throughput is more or less unchanged, and network throughput is down by 12.5%. The combined throughput of data coming from the network onto the disk increases by 6.5%; we believe this is caused by the performance isolation of running the disk and network drivers in separate VMs.

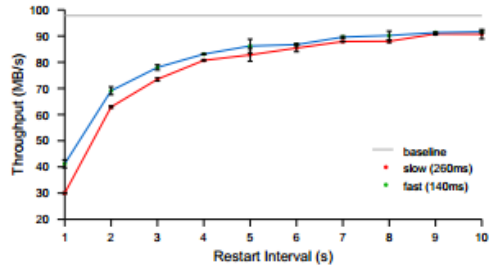

Figure 18: `wget` throughput while restarting *NetBack* at different time intervals.

To measure the effect of micro rebooting driver VMs, we ran the 2 GB `wget` to */dev/null* while restarting *NetBack* at intervals between 1 and 10 seconds. Two different optimizations for fast micro reboots are shown. In the first (marked as slow in Figure 18), the device hardware state is left untouched during reboots; in the second ("fast"), some configuration data that would normally be renegotiated via Xen Store is persisted. In "slow" restarts the device downtime is around 260 ms, measuring from when the device is suspended to when it responds to network traffic again. The optimizations used in the "fast" restart reduce this downtime to around 140 ms. Resetting every 10 seconds causes an 8% drop in throughput, as `wget`s TCP connections respond to the breaks in connectivity. Reducing the interval to one second gives a 58% drop. Increasing it beyond 10 seconds makes very little difference to throughput. The faster recovery gives a noticeable benefit for very frequent reboots but is worth less than 1% for 10-second reboots."

Performance and the fact that *Xen* has been adopted by many cloud service provider are in our view the main factors that prevent the widespread adoption of *XOAR*. Moreover, it

Xen has undergone significant updates.

Problem 9. *Discuss the impact of international agreements regarding privacy laws on cloud computing.*

Different countries and groups of countries such as the EU, have significantly different views on privacy. For example, EU privacy laws are considerably stricter than those adopted by the US. As cloud computing gains wider acceptance and public clouds offer significant cost savings and better user experience than other paradigms it becomes increasingly more important to have international agreements regarding data privacy.

As cloud service providers (CSPs) are likely to subcontract some of the services to organization in countries with low labor costs, a user is faced with uncertainties regarding the privacy of the data. Already CSPs such as Amazon, have data centers in countries on five continents; to offer the user some control over the data it owns, AWS requires a user to specify the region and the zone where his/her data should be stored and processed.

Data privacy in an example of a pervasive problem in our society, the fact that the law cannot keep up with the technology. The lack of international agreements on data privacy is already a serious problem that could threaten the future development of new technologies.

10 Chapter 10 - Complex Systems

Problem 1. *Search the literature for papers proposing a layered architecture for computing clouds and analyze critically the practicality of each of these approaches.*

There is some confusion regarding the term "layered cloud architecture." For example, in *http://texdexter.wordpress.com/2009/09/09/the-structure-of-the-new-it-frontier-cloud-computing-%E2%80%93-part-i/* the authors argue that: "Generally, the Cloud architecture defines 4 distinct layers from physical hardware to end user applications. Physical resources including spare desktop machines, clusters and datacenters form the lowest level of the stack, on top of which the virtual infrastructure is deployed." There are several so called "cloud layered architecture" models available on different Internet sites, e.g., the one in Figure 19.

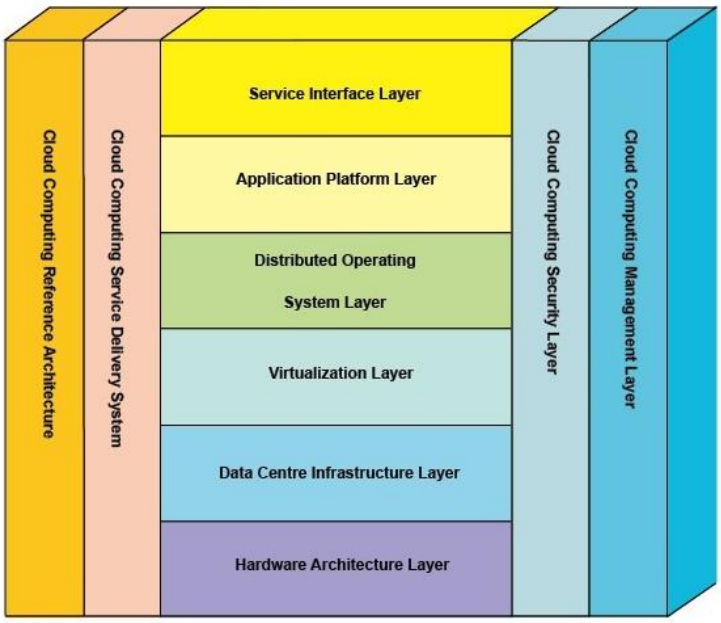

Figure 19: Yet another layered cloud architecture model.

Obviously, any system can be regarded as layered at some level of abstraction, but the common use of the term "layered" implies that the functions provided by the system are the result of interaction between system components organized in layers such that each layer interacts only with the adjacent ones. So layering implies modularity as a necessary but not sufficient condition. But layering requires a very restrictive communication patterns between the modules.

Modularity is a must for any large-scale system thus, a cloud software stack is based on modular design. Yet, it is very hard to imagine a functionally layered cloud architecture; indeed, there are many channels of interaction between the very large number of components of a cloud infrastructure, as we see in the NIST architectural reference model, Figure 20.

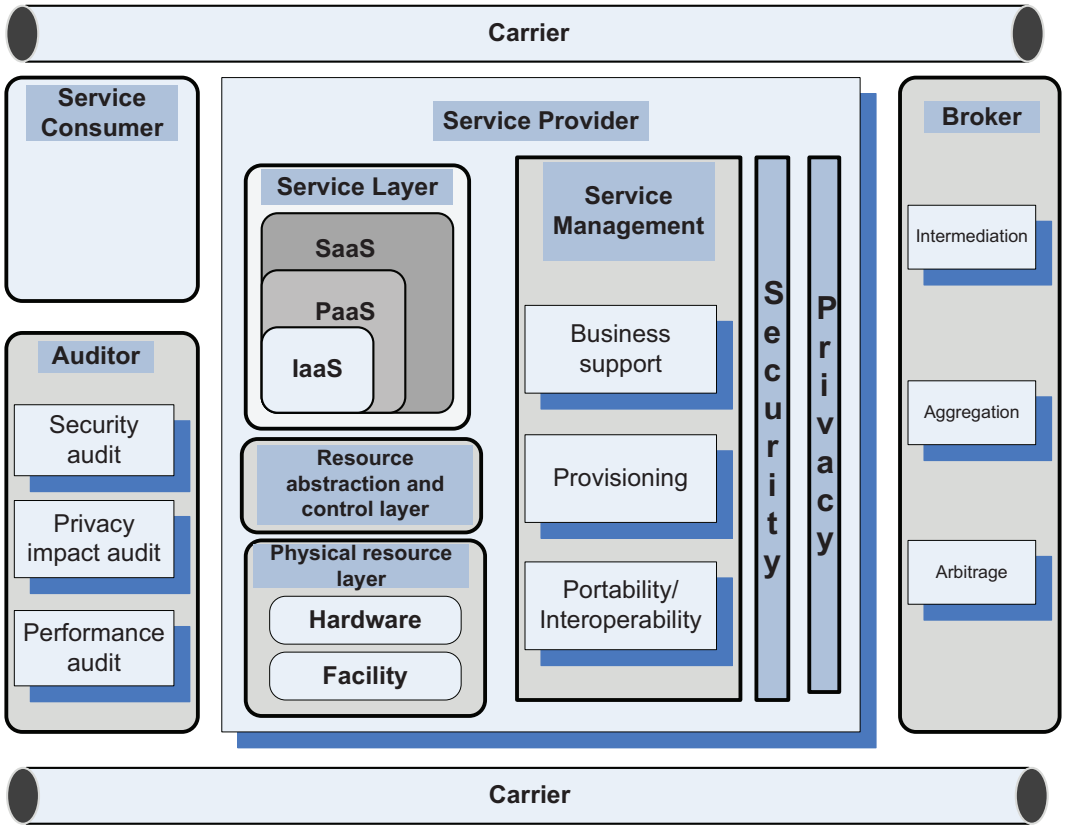

Figure 20: The entities involved in a service-oriented computing and, in particular, in cloud computing according to the NIST chart.

Problem 2. *Discuss means to cope with the complexity of computer and communication systems other than modularity, layering, and hierarchy.*

An obvious alternative is self-organization which often means also self-management and self-repair. Informally, self-organization means synergetic activities of elements when no single element acts as a coordinator and the global patterns of behavior are distributed [30, 70]. The intuitive meaning of self-organization is captured by the observation of Alan Turing [82]: "global order can arise from local interactions."

Self-organization is prevalent in nature; for example, in chemistry this process is responsible for molecular self-assembly, for self-assembly of monolayers, for the formation of liquid and colloidal crystals, and in many other instances. Spontaneous folding of proteins and other biomacromolecules, the formation of lipid bilayer membranes, the flocking behavior of different species, the creation of structures by social animals, are all manifestation of self-organization of biological systems.

Inspired by biological systems, self-organization was proposed for the organization of different types of computing and communication systems [40, 57], including sensor networks, for space exploration [39], or even for economical systems [50].

Self-organization requires individual entities to decide on their own the role they will play in the system; it also requires entities to make most of the decisions based on local state information whenever they are forced to respond to an external or internal event. Self-organization in a cloud computing environment is discussed in [64]. The main theme of [64] is that cen-

tralized control is not feasible in a large-scale system due to physical consideration such as large communication delay, inability of a central authority to respond promptly to requests from a very large number of entities. The only alternative is then a clustered organization. A scale-free network allows individual nodes to decide on the role they will assume based on the degree of the node; the few nodes of a high degree thus, connected to many nodes will assume a role of coordinator and nodes at minimum distance to the coordinator will join it to form a cluster. In such a cluster most resource management decisions can be made locally.

Problem 3. *Discuss the implications of the exponential improvement of computer and communication technologies on system complexity.*

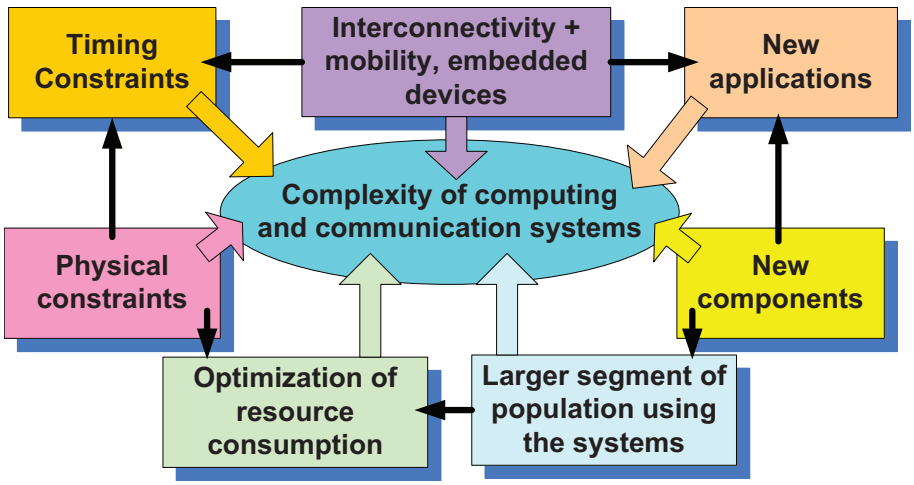

Figure 21: Factors contributing to the complexity of modern computing and communication systems. The slim black arrow show a causality relation between individual factors; for example, physical constraints demand optimization of resource consumption.

A number of factors contribute to the complexity of modern computing and communication systems, as shown in Figure 21; some of them are:

- The rapid pace of technological developments and the availability of relatively cheap and efficient new system components, such as multi-core processors, sensors, retina displays, and high-density storage devices.
- The development of new applications which take advantage of the new technological developments.
- The ubiquitous use of the systems in virtually every area of human endeavor which, in turn, demands a faster pace for hardware and software development.
- The need for interconnectivity and the support for mobility.
- The need to optimize the resource consumption.
- The constraints imposed by the laws of physics, such as heat dissipation and finite speed of light.

Each one of these factors could have side-effects. For example, the very large number of transistors on a chip allow hiding of *hardware Trojan horses (HTH)*[12], malignant circuits, extremely hard to detect. Wireless communication created new challenges for system security and increased security demands more complex software.

All the factors illustrated in Figure 21 are deeply related to the very fast pace of the technological developments; we see, in fact, a system with positive feedback, where each factor triggers a more intense contribution from all other factors.

Problem 4. *In Section 5.12 we discussed a side effect of abstractions, yet another method to cope with complexity. We showed how a VMBK (Virtual-Machine Based Rootkit) could pose serious security problems. Give other examples when abstraction of a physical system could be exploited for nefarious activities.*

An API abstracts the methods to interact with an application and the physical system it runs on. A malicious user or site could use any of the methods discussed in Chapter 9, e.g., phishing, SQL injection, or cross-site scripting, to attack legitimate sites.

Problem 5. *Give examples of undesirable behavior of computing and communication system that can be characterized as phase-transitions. Can you imagine a phase-transition in a cloud?*

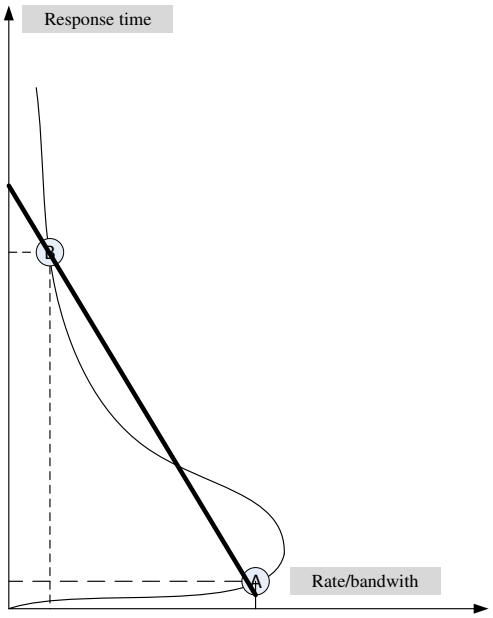

Figure 22: The quality versus quantity dependence for a system.

Figure 22 illustrates the behavior of many computer and communication systems due to a phase transition. The operating point moves rapidly from a state when the both the quality and the quantity of the function provided by the system are high to one when both an low.

[12]The HTH is said to be *functional* when gates are added to or deleted from the original design; a *parametric* HTH reduces the reliability of the chip.

For many computing and communication systems the quality is measured by the response time and the quantity is measured by the number of operations per unit of time, the rate, or the bandwidth of the system. A system can operate on point A on this characteristic curve, when the response time is low and the rate is high; a slight change in the environment or the system itself could trigger a rapid move to of the operation to point B where the quality measured by the response time is low and the quantity measured by the rate is also low.

This rapid transition is often triggered by congestion, the load placed on the system exceeds the capacity of the system. We see the effects of congestion on the transportation systems when we get stuck in traffic for hours. In a congested packet-switched network the routers start dropping packets, the senders timeout and resend the packets amplifying congestion. Very rapidly the system ends in a state when no traffic comes trough.

Similarly, in a time-shared system two processes paging intensively lead to trashing, no actual work is being done, the systems keeps fetching pages from the secondary storage and switching from one process to another.

Problem 6. *Can you attribute the limited success of the Grid movement to causes that can be traced to emergence?*

Emergence lacks a clear and widely accepted definition, but it is generally understood as *a property of a system that is not predictable from the properties of individual system components.* There is a continuum of emergence spanning multiple scales of organization.

Physical phenomena which do not manifest themselves at microscopic scales but occur at macroscopic scale are manifestations of emergence. For example, temperature is a manifestation of the microscopic behavior of large ensembles of particles. For such systems at equilibrium the temperature is proportional to the average kinetic energy per degree of freedom; this is not true for ensembles of a small number of particles. Even the laws of classical mechanics can be viewed as limiting cases of quantum mechanics applied to large masses.

Emergence could be critical for complex systems such as the financial systems, the air-traffic system, and the power grid. The May 6, 2010 event when the Dow Jones Industrial Average dropped 600 points in a short period of time is a manifestation of emergence. The cause of that failure of the trading systems is attributed to interactions of trading systems developed independently and owned by organizations which work together, but their actions are motivated by self interest.

A recent paper [76] points out that dynamic coalitions of software-intensive systems used for financial activities pose serious challenges, because there is no central authority and there are no means to control the behavior of the individual trading systems. The failures of the power grid (for example, the Northeast blackout of 2003) can also be attributed to emergence; indeed, during the first few hours of this event the cause of the failure could not be identified due to the large number of independent systems involved. Only later it was established that multiple causes, including the deregulation of the electricity market and the inadequacy of the transmission lines of the power grid, contributed to this failure.

The same arguments can be used to explain the limited success of the Grid movement: (i) interactions of systems developed independently and owned by organizations which work together, but their actions are motivated by self interest; (ii) dynamic coalitions of software-intensive systems pose serious challenges because there is no central authority and there are no means to control the behavior of the individual systems.

Problem 7. *Hardware Trojan horses (HTHs), malicious modifications of the circuitry of an*

integrated circuit, are very difficult to detect. What are, in your view, the main challenges for detecting HTHs? Discuss how different conceptual methods to detect HTH are impractical due to the complexity of the integrated circuits.

An HTH is very difficult to detect because the states when the system displays nefarious behavior can be reached only under very special circumstances thus, testing is unlikely to find the path to reach these states. It is very easy to hide a counter which triggers such nefarious activities in a processor with several million transistors. The power dissipation of the added or modified circuits is below any detectable threshold.

Problem 8. *Kolmogorov complexity seems to be a theoretical concept fairly remote from any practical application. Yet the simulation of a computer and/or communication system seems to be informally related to Kolmogorov complexity. Can you explain why?*

Complexity could be related to the description of a system and may consist of structural, functional, and, possibly, other important properties of the system. The Kolmogorov complexity $K_{\mathcal{V}}(s)$ of the string s with respect to the universal computer $\mathcal{V}$ is defined as the minimal length over all programs $Prog_{\mathcal{V}}$ that print s and halt

$$K_{\mathcal{V}}(s) = min[Length(s)] \text{ over all Prog: } \mathcal{V}(Prog_{\mathcal{V}}) = s. \tag{55}$$

The intuition behind Kolmogorov complexity is to provide the shortest possible description of any object or phenomena.

To simulate a system we have to describe all the states the system can be found in, as well as, the condition which trigger the transitions between states. All these elements are components of the system description. Moreover, this description must be as succinct as possible to be computationally feasible to simulate the system; this means that the number of states and the transitions between states should be the smallest set which allows an accurate modeling of the system.

Problem 9. *If you define the entropy of a system based on the probability of being in each of the states it can reach in the phase space. The phase space is an abstraction in which all possible states of a system are represented, with each possible state corresponding to one unique point in the space. The number of dimensions of the phase space of a system is given by the number of degrees of freedom of the system. Can you accept the uncertainty in the system, thus, the entropy, as a measure of system complexity? Can you justify why modularization decreases the system complexity?*

Our argument is based on the reduction of the cardinality of the state space of a modular system versus the monolithic version of the system. Indeed, a module exposes to other modules only a limited number of states, the states exposed by its interface, and it hides its internal states. In a model of the modular version of the system all undesirable internal states of a module can be aggregated in a single error state. This simplifies the design and verification of the system, as individual modules can be developed and tested separately and when a module lands in an error state it can be debugged independently of the other modules. A module can also be replaced with another module, as long as the two present the same interface to the other modules.

We can thus conclude that the entropy of a modular system is lower than the entropy of the monolithic version of the system and the fact that modularization simplifies the design

and verification of the system allows us to conclude that modularization decreases the system complexity.

11 Literature

[1] B. Addis, D. Ardagna, B. Panicucci, and L. Zhang. "Autonomic management of cloud service centers with availability guarantees." *Proc. IEEE 3rd Int. Conf. on Cloud Computing*, pp. 220–227, 2010.

[2] R. Albert, H. Jeong, and A.-L. Barabási. "The diameter of the world wide web." *Nature,* **401**:130–131, 1999.

[3] R. Albert, H. Jeong, and A.-L. Barabási. "Error and attack tolerance of complex networks." *Nature,* **406**:378–382, 2000.

[4] R. Albert and A-L. Barabási. "Statistical mechanics of complex networks." *Reviews of Modern Physics,* **72**(1):48–97, 2002.

[5] T. Andrei. "Cloud computing challenges and related security issues." *http://www1.cse.wustl.edu/jain/cse571-09/ftp/cloud/index.html*

[6] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwing, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. "Web Service Agreement Specification." *http://www.gridforum.org/Meetings/GGF11/Documents/draftggfgraap- agreement.pdf,* 2004.

[7] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. "Energy-aware autonomic resource allocation in multi-tier virtualized environments." *IEEE Trans. on Services Computing,* **5**(1):2–19, 2012.

[8] M. Asay. "An application war is brewing in the cloud." *http://news.cnet.com/8301-13505_3-10422861-16.html*

[9] J. Baker, C. Bond, J. C. Corbett, J. J. Furman, A. Khorlin, J. Larson, J.-M. Léon, Y. Li, A. Lloyd, and V. Yushprakh. "Megastore: Providing scalable, highly available storage for interactive services." *Proc. 5th Biennial Conf. on Innovative Data Systems Research (CIDR'11)*, pp. 223–234, 2011.

[10] M. Balduzzi, J. Zaddach, D. Balzarotti, E. Kirda, and S. Loureiro. "A security analysis of Amazon's elastic compute cloud service." *Proc. 27th Annual ACM Symp. on Applied Computing*, pp. 1427–1434, 2012.

[11] A-L. Barabási and R. Albert. "Emergence of scaling in random networks," *Science*, **286**(5439):509–512, 1999.

[12] A-L. Barabási, R. Albert, and H. Jeong. "Scale-free theory of random networks; the topology of World Wide Web." *Physica A*, **281**:69–77, 2000.

[13] N. F. Britton. *Essential Mathematical Biology.* Springer, 2004.

[14] M. Burrows. "The Chubby lock service for loosely-coupled distributed systems." *Proc. Symp. OS Design and Implementation (OSDI06),* pp. 335–350, 2006.

[15] C. Cacciari, F. D'Andria, M. Gonzalo, B. Hagemeier, D. Mallmann, J. Martrat, D. G. Perez, a. Rumpl, W. Ziegler, and C. Zsigri. "elasticLM: A novel approach for software licensing in distributed computing infrastructures." *Proc. IEEE 2nd Int. Conf. on Cloud Computing Technology and Science,* pp. 67–74, 2010.

[16] K. M. Chandy and L. Lamport. "Distributed snapshots: determining global states of distributed systems." *ACM Transactions on Computer Systems*, **3**(1):63–75, 1985.

[17] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. "Bigtable: a distributed storage system for structured data." *Proc. Conf. OS Design and Implementation* (OSDI06), pp. 205–218, 2006.

[18] R. Cohen and S. Havlin. "Scale-free networks are ultrasmall." *Phys. Rev. Lett.,* **90**(5):058701, 2003.

[19] L. Colitti, S. H. Gunderson, E. Kline, and T. Refice. "Evaluating IPv6 adoption in the internet." *Proc. Passive and Active Measurement Conference, PAM '2010*, pp.141–150, 2010.

[20] P. Colp, M. Nanavati, J. Zhu, W. Aiello, G. Coker, T. Deegan, P. Loscocco, and A. Warfield. "Breaking up is hard to do: security and functionality in a commodity hypervisor." *Proc. Symp. Operating Systems Principles,* pp. 189–202, 2011.

[21] H. A. David. newblock *Order Statistics,* Wiley, 1981.

[22] V. M. Eguiluz and K. Klemm. "Epidemic threshold in structured scale-free networks." *arXiv:cond-mat/0205439v.1*, 2002

[23] P. Erdös and A. Rényi. "On random graphs." *Publicationes Mathematicae*, **6**:290–297, 1959.

[24] P. Erdös and T. Gallai. "Graphs with points (vertices) of prescribed degree (Gráfok elört fokú pontokkal)." *Mat. Lapok* **11**:264–274, 1961; *Zentralblatt Math.* 103.39701.

[25] S. Floyd and Van Jacobson. "Link-sharing and resource management models for packet networks." *IEEE/ACM Trans. on Networking*, **3**(4):365–386, 1995.

[26] B. Ford. "Icebergs in the clouds; the other risks of cloud computing." *Proc. 4th Workshop on Hot Topics in Cloud Computing*, arXiv:1203.1979v2, 2012.

[27] M. Franklin, A. Halevy, and D. Maier. "From databases to dataspaces: A new abstraction for information management." *Sigmod Record,* **34**(4):27–33, 2005.

[28] S. Garfinkel and M. Rosenblum. "When virtual is harder than real: security challenges in virtual machines based computing environments." *Proc. Conf. Hot Topics in Operating Systems,* pp. 20–25, 2005.

[29] S. Garfinkel. "An evaluation of Amazon's grid computing services: EC2, S3, and SQS." *Technical Report, TR-08-07,* Harvard University, 2007.

[30] M. Gell-Mann. "Simplicity and complexity in the description of nature." *Engineering and Sciences*, LI, vol. 3, California Institute of Technology, pp. 3-9, 1988.

[31] S. Ghemawat, H. Gobioff, and S.-T. Leung. "The Google file system." *Proc. 19th ACM Symp. on Operating Systems Principles (SOSP'03)*, pps. 15, 2003.

[32] K. I. Goh, B. Kahang, and D. Kim. "Universal behavior of load distribution in scale-free networks." *Physical Review Letters,* **87**:278701, 2001.

[33] R. P. Goldberg. "Architectural principles for virtual computer systems." *Thesis, Harvard University*, 1973.

[34] P. Goyal, X. Guo, and H. M. Vin. "A hierarchial CPU scheduler for multimedia operating systems." *Proc. OSDI 96, Second USENIX Symp. on Operating Systems Design and Implementation*, pp. 107–121, 1996.

[35] J. Gray. "The transaction concept: virtues and limitations." *Proc. 7 Int. Conf. on Very Large Databases,* pp. 144–154, 1981.

[36] J. Gray and D. Patterson. "A conversation with Jim Gray." *ACM Queue* **1**(4):8–17, 2003.

[37] T. G. Griffn, F. B. Shepherd, and G. Wilfong. "The stable paths problem and interdomain routing." *IEEE/ACM Trans. on Networking,* 10(2):232–243, 2002.

[38] N. Gruschka and M. Jensen. "Attack surfaces: A taxonomy for attacks on cloud services." *Proc. IEEE 3rd Int. Conf. on Cloud Computing*, pp. 276–279, 2010.

[39] M. Hinchey, R. Sterritt, C. Rouff, J. Rash, W. Truszkowski. "Swarm-based space exploration." *ERCIM News* 64, 2006.

[40] J. Hopfield. "Neural networks and physical systems with emergent collective computational abilities." *Proc. National Academy of Science,* 79, pp. 2554-2558, 1982.

[41] IBM Smart Business. "Dispelling the vapor around the cloud computing. Drivers, barriers and considerations for public and private cloud adoption." White Paper, 2010. *ftp.software.ibm.com/common/ssi/ecm/en/ciw03062usen/CIW03062USEN.PDF.*

[42] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. Wasserman, N. J. Wright. "Performance analysis of high performance computing applications on the Amazon Web services cloud." *Proc. IEEE Second Int. Conf. on Cloud Computing Technology and Science,* pp. 159–168, 2010.

[43] H. Jin, X.-H. Sun, Y. Chen, and T. Ke. "REMEM: REmote MEMory as checkpointing storage." *Proc IEEE 2nd Int. Conf. on Cloud Computing Technology and Science,* pp. 319–326, 2010.

[44] E. Kalyvianaki, T. Charalambous, and S. Hand. "Self-adaptive and self-configured CPU resource provisionong for virtualized servers using Kalman filters." *Proc. 6th Int. Conf. Autonomic Comp. (ICAC2009),* pp. 117–126, 2009.

[45] E. Kalyvianaki, T. Charalambous, and S. Hand. "Applying Kalman filters to dynamic resource provisioning of virtualized server applications." *Proc. 3rd Int. Workshop Feedback Control Implementation and Design in Computing Systems and Networks (FeBid),* pps. 6, 2008.

[46] J. Kephart, H. Chan, R. Das, D. Levine, G. Tesauro, F. Rawson, and C. Lefurgy. "Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs." *Proc. 4th Int. Conf. Autonomic Computing (ICAC2007),* pp. 100-109, 2007.

[47] J. Kephart. "The utility of utility." *Proc. Policy 2011*, 2011.

[48] W. O. Kermack and A. G. McKendrick. " A contribution to the theory of epidemics." *Proc. Royal Soc. London, A*, **115**:700–721, 1927.

[49] A. N. Kolmogorov. "Three approaches to the quantitative definition of information." *Problemy Peredachy Informatzii*, **1**:4-7, 1965.

[50] P.R. Krugman. "The Self-organizing Economy." *Blackwell Publishers,* 1996.

[51] D. Kusic, J. O. Kephart, N. Kandasamy, and G. Jiang. "Power and performance management of virtualized computing environments via lookahead control." *Proc. 5th Int. Conf. Autonomic Comp. (ICAC2008)*, pp. 3–12, 2008.

[52] P. A. Loscocco , S. D. Smalley , P. A. Muckelbauer , R. C. Taylor , S. J. Turner , and J. F. Farrell. "The inevitability of failure: fhe flawed assumption of security in modern computing nvironments." newblock *Proc. 21 National. Inf. Sys. Security Conf*, pp. 303–314, 1998.

[53] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis,* McGraw-Hill, 1892.

[54] B.M. Leiner, V. G. Cerf, D. D. Clark, R. E. Khan, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts and S. Wolff. "A brief history of the Internet." *ACM SIGCOMM Computer Communicatiion Review*, **1**(5): 22 –39, 2009.

[55] E. Le Sueur and G. Heiser. "Dynamic voltage and frequency scaling: the laws of diminishing returns." *Proc. Workshop on Power Aware Computing and Systems, HotPower'10*, pp. 2–5, 2010.

[56] K. Ludwig, A. Keller, A. Dan, R. King, R. Franck. "Web service level agreement (WSLA) language specification." *IBM Corporation*, (2003)

[57] D. C. Marinescu, C. Yu, and G. M. Marinescu. "Scale-free, self-organizing very large sensor networks." Journal of Parallel and Distributed Computing (JPDC), **50**(5):612–622, 2010.

[58] M. W. Mayer. "Architecting principles for system of systems." *Systems Engineering,* **1**(4):267–274, 1998.

[59] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel. "Diagnosing performance overheads in Xen virtual machine environments." *Proc. First ACM/USENIX Conf. on Virtual Execution Environments,* 2005.

[60] A. Menon, A. L. Cox, and W. Zwaenepoel. "Optimizing network virtualization in Xen." *Proc. 2006 USENIX Annual Technical Conf.,* pp. 15–28, 2006. Also, *http://www.usenix.org/event/usenix06/tech/menon/menon_html/paper.html.*

[61] R. J. T. Morris and B. J. Truskowski. "The evolution of storage systems." *IBM Systems Journal*, **42**(2):205–217, 2003.

[62] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig. "Intel virtualization technology: hardware support for efficient processor virtualization." *Intel Technology Journal,* **10**(3):167–177, 2006.

[63] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G.Shin. "Performance evaluation of virtualization technologies for server consolidation." *HP Technical Report HPL-2007-59*, 2007lso, *http://www.hpl.hp.com/techreports/2007/HPL-2007-59R1.pdf*

[64] A. Paya and Dan C. Marinescu. "Clustering algorithms for scale-free networks and applications to cloud resource management." http://arxiv.org/pdf/1305.3031v1.pdf, May 2013.

[65] S. Pearson and A. Benameur. "Privacy, securitry, and trust issues arising from cloud computing." *Proc. Cloud Computing and Science,* pp. 693–702, 2010.

[66] M. Price. "The paradox of security in virtual environments." *Computer,* **41**(11):22–28, 2008.

[67] R. Rodrigues and P. Druschel. "Peer-to-peer systems." *CACM*, **53**(10):72–82, 2010.

[68] M. Rosenblum and T. Garfinkel. "Virtual machine monitors: Current technology and future trends." *Computer,* **38**(5):39–47, 2005.

[69] M. Satyanarayanan. "A survey of distributed file systems." *CS Technical Report, CMU*, *http://www.cs.cmu.edu/ satya/docdir/satya89survey.pdf*, 1989.

[70] P. Schuster. "Nonlinear dynamics from Physics to Biology. Self-organization: An 53 old paradigm revisited." *Complexity,* **12**(4):9-11, 2007.

[71] D. Sehr, R. Muth, C. Biffle, V. Khimenko, E. Pasko, K. Schimpf, B. Yee, and B. Chen. "Adapting software fault isolation to contemporary CPU architectures." *Proc. 19th USENIX Conf. on Security (USENIX Security'10),* pp. 1–11, 2010.

[72] P. Sempolinski and D. Thain. "A comparison and critique of Eucalyptus, OpenNebula and Nimbus." *Proc IEEE 2nd Int. Conf. on Cloud Computing Technology and Science,* pp. 417–426, 2010.

[73] S. Sivathanu, L. Liu, M. Yiduo, and X. Pu. "Storage management in virtualized cloud environment." *Proc. IEEE 3rd Int. Conf. on Cloud Computing*, pp. 204–211, 2010.

[74] J. E. Smith and R. Nair. "The architecture of virtual machines." *Computer,* **38**(5):32–38, 2005.

[75] L. Snyder. "Type architectures, shared memory, and the corolary of modest potential." *Ann. Rev. Comp. Sci.* 1, pp. 289–317, 1986.

[76] I. Sommerville, D. Cliff, R. Calinescu, J, Keen, T. Kelly, M. Kwiatowska, J. McDermid, and R. Paige. "Large-scale IT complex systems." *Communications of the ACM*, **55**(7):71–77, 2012.

[77] M. Stokely, J. Winget, E. Keyes, C. Grimes, and B. Yolken. "Using a market economy to provision compute resources across planet-wide clusters." *Proc. Int. Parallel and Distributed Processing Symp. (IPDPS 2009)*, pp. 1–8, 2009.

[78] M. Stonebreaker. "The "NoSQL" has nothing to do with SQL." *http://cacm.acm.org/blogs/blog-cacm/50678-the-nosql-discussion-has-nothing-to-do-with-sql/fulltext,* 2009.

[79] SUN Microsystems. "Introduction to cloud computing architecture." (White Paper, June 2009). *http://eresearch.wiki.otago.ac.nz/images/7/75/Cloudcomputing.pdf*

[80] Z. Toroczkai and K. E. Bassler. "Jamming is limited in scale-free systems." *Nature,* **428**:716, 2004.

[81] C. Tung, M. Steinder, M.!Spreitzer, and G. Pacifici. "A scalable application placement controller for enterprise data centers." *Proc. 16th Int. Conf. World Wide Web (WWW2007),* 2007.

[82] A.M. Turing. "The chemical basis of morphogenesis." *Philosophical Transactions of the Royal Society of London,* Series B 237:37-72, 1952.

[83] J. van Vliet, F. Paganelli, S. van Wel, and D. Dowd. "Elastic Beanstalk: Simple Cloud Scaling for Java Developers." O'Reilly Publishers. Sebastopol, California, 2011.

[84] K. Varadhan, R. Govindan, and D. Estrin. "Persistent route oscillations in interdomain routing." *Computer Networks,* **32**(1):1-16, 2000.

[85] D. J. Watts and S. H. Strogatz. "Collective-dynamics of small-world networks," *Nature,* **393**:440–442, 1998.

[86] A. Whitaker, M. Shaw, and S. D. Gribble. "Denali; lightweight virtual machines for distributed and networked applications." *Technical Report 02-0201,* University of Washington, 2002.