From a Virtualized Computing Nucleus to a Cloud Computing Universe: A Case for Dynamic Clouds

Divyakant Agrawal Sudipto Das Amr El Abbadi
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106 - 5110, USA
{agrawal, sudipto, amr}@cs.ucsb.edu

ABSTRACT

The current model of the cloud consists of a *static* set of data centers (or **cloud cores**) which drive the computation and storage needs of large numbers of applications. We envision a new paradigm where the cloud will be comprised of a large *dynamic* collection of cloud cores along with a static set of cores, the *nucleus*, to create a cloud computing universe with a capacity much larger than the nucleus and a cost much smaller than owning the entire infrastructure. This model is rooted by the observation that a tremendous amount of computation exists outside the core that can potentially augment the nucleus' capacity. An example of this surplus capacity are enterprizes with diurnal trends in usage behavior that join the cloud during *predicted* periods of usage troughs. We propose to leverage this elastic and dynamic infrastructure to create a unified cloud service. A number of challenges, at all levels of the software stack, need to be addressed for these futuristic architectures to become a reality. We focus on the challenge of an *elastic* and *agile* data management infrastructure to deal with the dynamics associated with this novel paradigm.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Design

Keywords

Dynamic clouds, database systems, elasticity, fault-tolerance, reliability

1. INTRODUCTION

Current cloud infrastructures are limited to a small set of geographically separated large data centers, **cloud cores**, representing a closely knit **nucleus**, supporting the computation and data storage requirements of large numbers of applications. These huge cloud cores are custom built to provide the economies of scale for operation and management—a major contributing factor to the success of the cloud. However, limiting computation to the nucleus prevents the cloud from leveraging the huge pool of resources available outside the nucleus. We envision a new trend in the cloud infrastructure which deviates from this model of a small static collection of computing cores to *many* (and often heterogenous) *loosely coupled* cloud cores, a paradigm we call **dynamic clouds**.

This new trend originates from the possibility of exploiting large amounts of computation and storage resources that exist outside the nucleus. For instance, some Wall Street financial companies have expressed the intent of "renting out" their resources which are idle during non-business hours [13]. Such diurnal usage trends, comprising periods of high usage followed by periods of zero or low usage, is also observed in other financial sector companies across the globe, as well as in infrastructures that serve businesses that do not operate 24×7. These companies need to own such huge infrastructures to sustain their core businesses, but lay idle during non-business hours. This presents the case for *dynamic clouds*, i.e. a cloud infrastructure with a small number of static cores (the nucleus) and a dynamic collection of cores that coalesce into the cloud for certain periods, while being unavailable for other periods. This paradigm leverages the predictable pattern of cores joining and leaving the cloud (*churn*) to constitute the dynamic cloud infrastructure.

This dynamic clouds model provides economic incentives for both the *cloud providers* and the companies renting the infrastructure (*cloud lessors*). The cloud providers can *rent* the dynamic cores at a fraction of the cost of owning and maintaining them, thus allowing them to increase their profit margins. On the other hand, the cloud lessors gain revenue for resources which they have to maintain for their core businesses, but lay idle outside their business hours. This creates a *trade* between the two involved entities. Our vision is to *integrate the static cloud nucleus and a large pool of dynamic cloud cores to create a cloud computing universe with a capacity much larger than the nucleus and a cost much smaller than owning the entire infrastructure*.

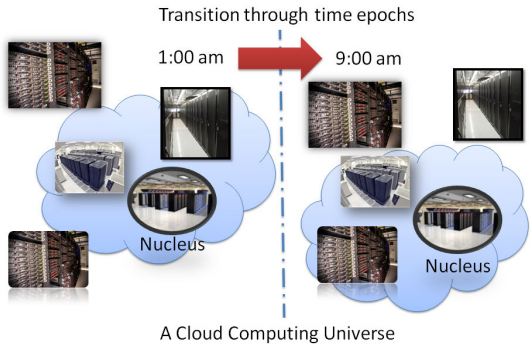

Figure 1: A dynamic cloud formed as a collation of a static nucleus (enclosed in the ellipse) and a dynamic collection of cores that together form the cloud.

Figure 1 provides an illustration where a nucleus combines with a dynamic collection of cores to form the cloud service. The static nucleus coordinates and synchronizes the dynamic cores, while the dynamic cores augment the capacity of the system. This paradigm

also presents the potential for small sized cloud providers to establish a large cloud service without owning a large infrastructure. In the years to come, dynamic clouds will present a radically different paradigm for cloud computing and will pose new challenges.

A large number of challenges at different levels of the software stack must be addressed to provide a common uniform infrastructure and automated integration of these dynamic cores into the cloud. These challenges include, and are not limited to, the need for *elastic* and *agile* data and application platforms, application traffic and load management amongst heterogenous infrastructures connected through wide area networks, tools for monitoring and autonomous management, ensuring privacy and security of data, efficient inter data center state synchronization, as well as other non-technical challenges such as legal and policy issues pertaining to data ownership or region specific statutory jurisdictions.

Our focus is the challenge of providing an elastic, agile, scalable, and fault-tolerant data management infrastructure for this new cloud paradigm. We discuss the design principles and suggest the design of some of the main building blocks of the infrastructure. Our proposed design separates the critical meta information of the system (called *system state*) from the application specific data and information (called *application state*). We propose the use of strong consistency protocol [12] for system state maintenance within a single cloud core, while using causal consistency [11] for loose coupling and synchronization between cloud cores.

This paper intends to start the discussion on the set of techniques and abstractions for this novel paradigm of dynamic clouds. A series of technological innovations are necessary to realize this transformation from a virtualized computing nucleus to a cloud computing universe; this paper takes the first steps in that direction.

2. DYNAMIC CLOUDS

We ground our discussion on dynamic clouds on some assumptions about the system model which will focus our presentation, but can be relaxed in the future: (i) we focus on the platform as a service (PaaS) abstraction where the cloud provider manages the tenant's data and application state and is responsible for load balancing and on-demand scaling; (ii) we assume a shared cloud data platform where each application's database (**tenant**) is being served from a single data center; (iii) we consider OLTP workloads characterized by short transactions with variable distribution of reads and writes; (iv) we assume that the availability periods of the dynamic cores are predictable or known in advance; (v) we assume that appropriate virtualization technologies can efficiently switch the resources of the dynamic cores from their regular business role to the role of being part of the cloud infrastructure; and (vi) we assume that the dynamic cores have state persistence so that when a dynamic core joins the cloud, it remembers its state from the last period it was part of the cloud.

2.1 Design Principles

In [1], we analyzed a number of scalable data management systems (such as Bigtable [3]) to identify important design principles that can be carried over to other scalable and elastic cloud database systems. This paper uses two of these design principles.

Separate System and Application State. The *system state* is the system metadata (such as catalog information or constituent nodes in a large loosely couple distributed system) critical to ensure the correct operation of the system. On the other hand, application specific data stored by the system is called the *application state*. The system state is typically multiple orders of magnitude smaller than the application state and has more stringent consistency and durability requirements compared to application state. Therefore, a clean separation allows using different sets of protocols for managing the different types of states.

Decouple Data Control from Data Storage. Control refers to the exclusive rights to access and update data. Our design leverages a clean separation between the *data control layer* and the *data storage layer*. Separating the physical data storage from the access logic has multiple benefits. *First*, the data storage layer can independently make data placement and replication decisions thus providing a fault-tolerant distributed network addressable storage abstraction. *Second*, the data control layer and the storage layer can independently scale depending on the requirements at each layer. *Third*, the data control layer can perform lightweight transfer of ownership to allow effective migration.

2.2 Technical Challenges

We now discuss some critical technical challenges that must be addressed to manage a cloud substrate across a collection of dynamic cores.

A uniform namespace for the dynamic cores. To provide a uniform view of a cloud service spanning a dynamic collection of cloud cores, a namespace spanning the cores is essential. This namespace comprises the system state for this dynamic cloud. Maintaining this state across all the cores and providing consistency guarantees for efficient and safe operation is critical.

Consistency models and abstractions. Ensuring data consistency is crucial for the correctness of the system. Strong consistency is expensive and eventual consistency is hard to reason about. In such a scenario, a challenge remains on devising appropriate consistency models that allows scaling to large numbers of nodes and cores, efficiently handles the underlying dynamics, while providing abstractions to allow the applications to reason about correctness.

Efficient integration of surplus capacity. The software substrate of a dynamic cloud consists of a number of layers. Techniques to efficiently synchronize and integrate these layers across the multiple cloud cores is essential. This synchronization allows cores to dynamically join and leave the cloud without affecting the operation of the larger cloud.

Effective load and data migration techniques. A major enabling feature of this dynamic cloud paradigm is elasticity and ability of the software layers to leverage the elasticity and dynamics in the underlying hardware layer. Effective techniques to balance load across the available cores, migrate application and data as mandated by the load balancing policies, and elastic data management are therefore critical.

Efficient state replication and replica placement. Since all dynamic cores are not available at all times, replication of system and application state is essential for both fault-tolerance and high availability. Using synchronous replication across geographically separated data centers will be prohibitively expensive. It is therefore critical to design consistency models for asynchronous replication. Furthermore, intelligent replica placement techniques are also important to handle the churn while ensuring high data availability and minimizing the amount of data transfer.

Large scale monitoring and system modeling. Monitoring the resources at the different cores, collecting usage statistics, and modeling behavior of the overall system form a critical component. A system controller uses these statistics and models to determine which tenants are hosted on which cores, and which tenants must be migrated as new capacity become available.

Data security and privacy. Like any shared infrastructure, a major challenge is to ensure data security and privacy, both for the cloud lessors as well as the tenants served by the cloud provider. In addition to the economic incentives, proven security guarantees are

critical to provide confidence to the lessors to rent out their infrastructure without compromising its integrity.

3. BUILDING BLOCKS

3.1 System Architecture

We propose that the software substrate for the data management system for a dynamic cloud consist of four layers: the *system monitoring and control layer*, the *metadata management layer*, the *data control layer* (for instance the database engines serving the data), and the *data storage abstraction*. Each cloud core has a local instance of the layers with a loose coupling of the layers across the cores. Considering the scale of tens of geographically separated cloud cores, interesting tradeoffs between consistency, performance, and availability arise. A single consistency model is not enough to characterize the different layers of a dynamic cloud. We therefore use a collection of consistency models for different layers of the system. This allows reasoning about its correctness, while minimizing the impact on performance and availability.

3.1.1 System Monitoring and Control Layer

The system controller is responsible for meta level control operations. At every core, the system controller's major responsibilities include monitoring system and recovering from server failures, gathering usage statistics and modeling load and system behavior, coordinating state synchronization to facilitate joining of a new core into the cloud, replica placement decisions, load balancing and tenant migration decisions both at the data control and data storage layers, and facilitating a core's departure and checkpointing its state. The controller uses its local view of the global metadata for its decisions.

3.1.2 Metadata Management Layer

The metadata manager, or the naming service, provides a unified view of the dynamic cloud and maintains the *system state*. This information is critical for correct operation of the system and is therefore replicated. Within a core, system state requires *strong consistency*. We propose to use the Paxos [12] protocol to consistently maintain this metadata within a core. Across cores, we propose a global naming service as a collection of these local naming services which merges the local metadata views. Straightforward merging of all the local views can give rise to inconsistencies, especially in an asynchronous environment where the communication sub-system can reorder the messages. On the other hand, strong consistency of global metadata is expensive, both in terms of performance and in terms of availability in the presence of failures.

We propose to create a global view over multiple local views by drawing upon the classical notion of *causality* [11] that has been studied extensively in distributed systems. In particular, we propose to maintain the local states as a distributed dictionary [2,7,18]. Each core maintains time tables corresponding to its knowledge about the system state of other cores. These time tables are exchanged during message transfers between the metadata managers at the different cores, and are in turn used to merge the states. This establishes a causal relationship and ensures that applications observe the state of the system which is *causally consistent*. When we combine the fact that no conflicting and competing decisions will be made by the local metadata manager at the different cores, *causal consistency* of the global metadata view is enough to guarantee consistent metadata maintenance. We underscore the relative elegance of our design since the notion of *causal consistency* is a natural notion of correctness in distributed systems as we transition from a synchronous environment to an asynchronous environment. Furthermore, implementations such as ISIS [2] have demonstrated that causality introduces relatively low performance overhead. Considering the similarity of dynamic clouds with P2P systems, techniques from P2P literature [16] might also present interesting approaches for metadata management.

3.1.3 Data Control Layer

The data control layer is responsible for handling the requests for accessing the tenant databases. It is essentially a database engine executing the tenant's transactions. The metadata layer is used as a catalog to direct a tenant's request to the server currently serving the tenant's data. One of the major requirements is that the data control layer must be lightweight and agile to allow inexpensive tenant migration. We propose an extension of the design of ElasTraS [5], a multitenant transactional database for the cloud. It consists of a collection of transaction managers (TM), where each tenant is exclusively served by a single TM, and each TM serves multiple tenants. A rich literature of efficient transaction processing techniques can be used to execute transactions at the TMs [17]. Fault tolerance and high availability is guaranteed by storing the entire application state, including the transaction logs, in the data storage layer.

3.1.4 Data Storage Layer

The data storage layer provides an abstraction of a fault-tolerant storage with block level access granularity. Such a storage abstraction is common in current clouds (e.g. the Google file system [9]) and supports strongly consistent access within a cloud core. But in the model for dynamic clouds, this unified storage layer conceptually spans multiple cloud cores. Therefore, strong replica consistency in the storage layer is not feasible. In our model, at any instant of time, a tenant is served from a single cloud core. Replication is therefore primarily used for fault-tolerance and to aid tenant migration between cloud cores. In such a scenario, the major benefit of strong replica consistency is that the permanent failure of the primary does not lead to data loss; but this benefit does not outweigh the performance penalty of strong consistency. We therefore propose timeline consistency [4] for the data storage layer with a timeline per tenant. Timeline consistency ensures that all replicas of a tenant's data see all updates in the same order. PNUTS [4] demonstrates the use of a guaranteed delivery publish-subscribe system for low cost cross-data center replication at web-scale. The primary replica commits the updates to the pub-sub system and the replicas subscribe to the updates. We propose a similar design to replicate data across the cloud cores. Metadata of the storage layer (i.e. mapping of the blocks to the cloud cores and the servers) is managed by the metadata management layer. Control decisions such as replica placement, re-balancing, and data migration are handled by the controller.

3.2 Agility and Elasticity

Agility and elasticity are the two most critical features for the effective use of a dynamic cloud infrastructure. The efficient handling of churn and effective migration and load balancing to use the surplus capacity and tolerate the churn are its two important components.

Handling churn. In the dynamic cloud model, churn of cores is predictable. Therefore, the system takes special measures to handle cores joining and leaving the cloud. Recall that the dynamic cores have persistent storage remembering its state. Let C_l denote the core leaving the cloud. C_l's controller initiates migrating access control for its share of tenants to other active cores. Once migration in the access layer is complete, C_l is now only responsible for the

application state in the storage layer. The controller checkpoints the metadata and the state of the data layer (stores a per tenant log sequence number for the last update seen). This checkpoint is used to facilitate the core rejoining the cloud and is replicated to a set of active cores. An almost reverse process is initiated when a core rejoins the cloud. Let C_j denote the core which is rejoining. C_j's controller uses its checkpoint prior to leaving to determine the incremental updates since previous checkpoint that must be transferred to synchronize the storage and metadata layers. The per-tenant timelines and the log based replication used in the data storage layer allows efficient synchronization. This synchronization is followed by migration of data ownership and access rights of some tenant applications to core C_j. The cost of a core leaving and rejoining the cloud is amortized over the time the core remains a part of the cloud.

Load balancing and elasticity. Migrating tenant applications and databases is an important primitive for effective load balancing and elasticity. The data storage layer replicates a tenant's data across multiple cloud cores. The controller selects which tenant to migrate and the destination core. Though the controller can select any core, selecting a core with a local copy minimizes data movement across cores. We propose the use of extensions to existing techniques for live database migration [6]. We leverage the causal communication in the metadata layer to coordinate this migration. To migrate control of a tenant T from core C_a to core C_b, C_a's controller inserts a *transfer(T, C_a, C_b)* event in the local metadata view of C_a and sends this message to the controller at C_b. C_a stops processing any application requests against T. Advanced techniques, as discussed in [6], can be used to minimize the availability window of T. A handover protocol is executed to ensure correct control transfer. C_b starts serving T on successful completion of the handover phase. The migration process terminates with the receipt of the $ACK\ \langle transfer(T, C_a, C_b)\rangle$ event at C_a. Other cores know about this migration through causal message transfers from C_a and C_b. The correctness of application level operations can be established since transfer events across multiple clouds are causally consistent.

3.3 Data Placement

Since all cores are not available at all time periods, prudent replica placement is essential to ensure high availability of a tenant's data. If n denotes the total number of cores, k denote the minimum number of active replicas ($k < n$), then the controller selects k' cores ($k < k' \leq n$) to replicate a tenant's data. Selection of the appropriate cores is determined by a model that accounts for the churn and load behavior. The overall goal of the placement algorithm is to minimize data movement across cores.

3.4 Data Security and Privacy

Ensuring the security and privacy of data hosted in a shared environment has been a long standing challenge [8]. In a static cloud, there must be a mutual trust between the cloud provider and the tenants to ensure that data is secure against various vulnerabilities [15]. Additionally, the research community has developed techniques such as information dispersal [14] or query execution on encrypted data [10], and many proprietary technologies that protect the current cloud deployments. From the security perspective, a dynamic cloud faces a set of challenges similar to that of a static public and hybrid cloud infrastructures. Ensuring data privacy continues to remain an active field of research and the new paradigm can leverage from the advances made in this area.

4. DISCUSSION AND FUTURE DIRECTIONS

We presented a preliminary design of a data management system to enable effective use of dynamic clouds. A number of discussed components present interesting directions for future work; we conclude with a discussion of a few other interesting directions.

Differential pricing. Depending on the availability and demand of resources, a variable cost can be associated with the cloud resources. For instance, a large number of cores might be available during the weekends, thus allowing lower priced resources. This allows the possibility of differential pricing and a marketplace for resource usage following different economic models to maximize profit. This differential pricing relates to Amazon's spot instances.

Interplay of Service Level Agreements (SLAs) and Cost of service. Cloud and multitenant systems charge for the SLAs they provide. Interesting tradeoffs arise on the kind of SLAs that can be supported in the dynamic cloud model. Can weaker SLA's be used to minimize the cost?

Supporting a diverse set of applications and workloads. We only discuss techniques to support update intensive web-applications and an OLTP database to drive these applications. For widespread adoption and success of this model, it is important to be able to support a wide variety of applications and workloads. A characterization of the feasible workloads and the challenges that arise for dealing with a variety of workloads is also important.

Acknowledgments

The authors would like to thanks Aaron Elmore, Shoji Nishimura, and Shiyuan Wang for their comments on earlier drafts of the paper. This work is partly funded by NSF grants III 1018637 and CNS 1053594.

5. REFERENCES

[1] D. Agrawal, A. El Abbadi, S. Antony, and S. Das. Data Management Challenges in Cloud Computing Infrastructures. In *DNIS*, pages 1–10, 2010.

[2] K. P. Birman and T. A. Joseph. Reliable communication in the presence of failures. *ACM Trans. Comput. Syst.*, 5(1):47–76, 1987.

[3] 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. In *OSDI*, pages 205–218, 2006.

[4] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. PNUTS: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1(2):1277–1288, 2008.

[5] S. Das, S. Agarwal, D. Agrawal, and A. El Abbadi. ElasTraS: An Elastic, Scalable, and Self Managing Transactional Database for the Cloud. Technical Report 2010-04, CS, UCSB, 2010.

[6] S. Das, S. Nishimura, D. Agrawal, and A. El Abbadi. Live Database Migration for Elasticity in a Multitenant Database for Cloud Platforms. Technical Report 2010-09, CS, UCSB, 2010.

[7] M. J. Fischer and A. Michael. Sacrificing Serializability to Attain High Availability of Data in an Unreliable Network. In *PODS*, pages 70–75, may 1982.

[8] M. Gertz and S. Jajodia. *Handbook of Database Security: Applications and Trends*. Springer, 2007.

[9] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *SOSP*, pages 29–43, 2003.

[10] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD*, pages 216–227, 2002.

[11] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[12] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[13] R. Miller. Wall street's cloudy opportunity. `http://goo.gl/2I3o`, April 2010.

[14] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36:335–348, April 1989.

[15] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *CCS*, pages 199–212, 2009.

[16] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[17] G. Weikum and G. Vossen. *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*. Morgan Kaufmann Publishers Inc., 2001.

[18] G. T. Wuu and A. J. Bernstein. Efficient solutions to the replicated log and dictionary problems. In *PODC*, pages 233–242, 1984.