A Model for Characterizing the Scalability of Distributed Systems

A Vijay Srinivas and D Janakiram
{avs, djram@cs.iitm.ernet.in}
Distributed & Object Systems Lab,
Dept. of Computer Science & Engg.,
Indian Institute of Technology, Madras, India
http://dos.iitm.ac.in

Abstract

Scalability is an important issue in the construction of distributed systems. A number of theoretical and experimental studies have been made on scalability of distributed systems. However, they have been either studies on specific technologies or have studied scalability in isolation. The main conjecture of our work is that scalability must be perceived along with the related issues of availability, synchronization and consistency. In this context, we propose a scalability model which characterizes scalability as being dependent on these factors as well as the workload and faultload. The model is generic and can be used to compare scalability of similar systems. We illustrate this by a comparison between NFS and AFS, two well known distributed file systems. The model is also useful in identifying scalability bottlenecks in distributed systems. We have applied the model to optimize Virat, a wide-area shared object space that we have built.

1 Introduction

Building large scale distributed systems is an arduous task. Unpredictable network latency, network failures, node failures and most importantly, absence of a global clock present difficulties in terms of access to shared data as well as knowledge of global progress. Some scalable distributed systems have been built, for example Google [1], Andrews File System (AFS) [2], Globe [3] and Astrolabe [4]. Each system has addressed specific aspects of scale. For example, AFS uses whole block caching as a key mechanism for building scalable file systems. Globe considers replication and caching as central issues for building scalable middleware. Astrolabe is a distributed information system that uses hierarchical location management as well as publish-subscribe architecture as the main features. However, there are no general design principles that have been abstracted out. The other key difficulty in large scale systems is the lack of characterization of scalability.

A number of studies have been made considering the scalability of distributed systems. A scalability metric based on productivity was defined in [5]. The metric evaluates scalability as the product of throughput and response time (or any value function) divided by the cost factor. An analytical model of server systems based on Layered Queuing Networks (LQN) was proposed in [6]. A more general analytical model of client-server systems based on rendezvous networks was given in [7]. It illustrates how certain slow servers could become bottlenecks and suggests threading/cloning to relieve these bottlenecks. Recent work on design patterns for concurrent and networked systems documents patterns such as *Reactor, Half-Sync Half-Async* for efficient multi-threading in server systems [8]. Scalability of Java 2 Enterprise Edition (J2EE) technology has been evaluated in [9]. The paper distinguishes between scale-out, which refers to adding more servers and scale-up which refers to adding more CPUs for each server. It concludes that for J2EE technology, scale-out is better.

All the above efforts perceive scalability in isolation and some are even specific to certain technologies. The inter-related issues of synchronization, consistency and availability are neglected. One of the conjectures of this thesis is that availability, consistency, synchronization and scalability are closely related and should be looked at in totality, not in isolation. For instance, it is well known that replication improves availability. However, if consistency requirements are strict, scalability and availability have to be traded-off. Brewer's conjecture [10] postulates that in the presence of network partitions, both consistency and availability cannot be achieved. Availability has been quantified in [11] and its trade-off with consistency has been studied. However, both [10] and [11] do not address the scalability issue.

We propose a scalability model that considers scalability as a function of synchronization, consistency, availability, workload and faultload. The model complements LQN based scalability analysis methods. These LQN based models can be used to arrive at upper bounds on the scalability achievable in a system, given workload and environment model. However, they do not analyse related factors which influence scalability. The proposed scalability model helps to identify bottlenecks in a distributed system and hence provides directions to improve the scalability of existing systems.

The rest of the paper is organized as follows. Section 2 provides a brief outline of Virat, the shared object space that we have built. Section 3 explains the essence of the scalability model proposed in the paper. Section 4 provides a comparison of two common file systems, Network File System of Sun and the Andrew File System of CMU, by applying the scalability model. Section 5 concludes the paper and provides directions for future research.

2 Virat: An Internet Scale Shared Object, Event and Service Space

It is non-trivial to extend existing Distributed Shared Memory (DSM) systems to the Internet scale. The additional messaging latencies, non-deterministic message deliveries and failures pose challenges. Specifically, DSMs need to use some form of checkpointing to recover from failures. Coordinated checkpointing involves synchronization between the various nodes to ensure consistent global states. Most existing DSMs use coordinated checkpointing which would inhibit their scalability [12]. Further, existing DSMs either use centralized mechanisms or some form of group communication (multicasting or broadcasting) for object location. This is another factor that affects their scalability. In this context, we have developed Virat, a wide area shared object space[1] that addresses these issues and provides a higher level of abstraction to application developers [15].

Virat supports middleware services such as naming and trading as well as replica object management. Virat uses an independent check pointing and lazy reconstruction mechanism to handle failures of object repositories. The object repositories (one per cluster) are responsible for cluster level management of replicas. Communication between the object repositories themselves is through a peer-to-peer protocol. This is useful for locating objects across clusters.

Virat provides a data centric Concurrency Control (CC) mechanism to realize various consistency models. Virat has been extended to a fully typed shared event space, facilitating publish-subscribe kind of interactions and large scale event notifications [16]. A shared service space has also been built over the shared object abstraction of Virat. This allows services to be discovered at runtime and composed, as well as dynamically reconfigured. Virat has been implemented using J2EE over an Institute wide network and its scalability is being tested over a WAN.

3 The Scalability Model

$scalability = f(avail, sync, consis, workload, faultload)$

- *avail* is availability - can be quantified as the ratio of number of transactions accepted versus submitted. Availability itself is a function of network availability (number of operations reaching any replica) and service availability (number of operations accepted by a replica).
- *consis* is consistency, itself a function of update ordering and consistency granularity. Update ordering refers to update ordering mechanisms across replicas and can be one of causal, serializable or PRAM. Consistency granularity refers to the grain size at which consistency needs to be maintained in the system. There are two

[1]A shared object space such as Linda [13] enables sharing at the level of application objects, unlike traditional Distributed Shared Memory (DSM) systems that share pages [14].

dimensions to consistency granularity. Caching is an important and well known method for improving performance. The size of caching is an important factor governing the performance. A higher size implies more chances of program locality effects and better performance. The other dimension is the granularity of consistency maintenance; whether it is at the individual replica level or using some form of aggregation. It may be easier to maintain consistency at aggregate level than at individual replica level, but at the cost of strictness.

- *sync* refers to synchronization among the replicas. The two dimensions of synchronization are how often the replicas are synchronized and the mode of synchronization. The former is captured in terms of the δ value [17], δ referring to the number of updates that can be buffered in a particular node before synchronizing with others. The latter can be modeled as a γ value, with $\gamma = 0$ implying a server push/invalidation model and a positive non-zero value indicating that a replica could miss some updates and can be updated lazily, either in push or pull mode. A similar idea has been exploited in the notion of *local consistency* [18], but without any attempt at quantification.

- Workload can be broken down into workload intensity and workload service demand characterization. Workload intensity refers to number of transactions per second or number of clients etc. and service demand refers to CPU time for operations, network delays etc.

- Faultload is the failure sequences and number as well as location of replicas.

The scalability model has been applied to Virat and it gives directions for optimization of Virat. First, relating to the consistency parameter, Virat has been optimized to implement different update ordering mechanisms. These imply different consistency mechanisms, namely causal consistency, sequential consistency and causal serializability. This is in line with the established principle that relaxing consistency ordering leads to improved scalability [18]. A further optimization in Virat relates to consistency granularity. Consistency is maintained at the level of object repositories only, and they are responsible for update propagation to other nodes. Thus, serializability protocol, if required is enforced only among the repositories and not among all replicas. These two optimizations improve the scalability of Virat.

3.1 Open Issues

The function f could be a mathematical function that can be used to quantify the scalability of a distributed system. At a higher level, the f can also be viewed as an algorithm. The primary goal of this algorithm is to identify scalability bottlenecks in the system. An open issue is whether the arguments to f are complete and minimal.

Another critical issue is the influence of architecture or system structure on scale. The kind of communication style and the structure that is dictated by the architecture would also influence the scalability. For instance, event based communication style might scale up better due to its inherent asynchronous nature. This is not currently captured, as there is no clear mechanism to quantify this influence.

4 Comparing NFS and AFS using the Scalability Model

A comparison can be made between two similar systems by applying the proposed scalability model. Network File System (NFS) of Sun [19] is a well known file system that is based on the Remote Procedure Call (RPC) paradigm. Andrews File System (AFS) has been proposed and realized in Carnegie Mellon University [20]. AFS claims better scalability compared to NFS due to its stress on client side computation resulting in decreased server loads.

Applying the proposed model to AFS, one can make out that there are no specific requirements or constraints on availability. An important design decision in AFS2 was to sacrifice UNIX semantics for file updates and provide only session semantics; it is a much less stricter guarantee, in that updates are buffered and made visible to other clients only on file closing. This implies that the δ value depends directly on session length. As long as a file is open and not invalidated by the server, updates are buffered. Updates are propagated to other clients only when a file is closed. AFS uses a cache invalidation based callback model for update propagation. This implies that any updates at the server results in callbacks to registered clients, resulting in a γ value of 0 in the model. AFS uses whole file caching, so the granularity of consistency is higher compared to the use of block caching. Studies have shown that the workload in AFS is characterized by temporal locality [21].

NFS also uses a delayed write protocol, but the δ value is higher compared to AFS due to the 3 second interval; the time after which all cache blocks become invalidated automatically. Secondly, NFS uses block caching, not whole file caching as in AFS. This implies that if the consistency granularity value in AFS is say λ, then for NFS, the value is $\frac{\lambda}{n}$ where n is the number of blocks in a file (ratio of file size to block size). Hence, to summarize the comparison, availability, faultload being similar, the scalability of AFS and NFS is dependent on three key parameters; consistency granularity, synchronization and workload characterization.
$scalability_{AFS} = f(\delta_{session}, \lambda, workload_{tempLoc})$
$scalability_{NFS} = f(\delta_{3sec\_bound}, \frac{\lambda}{n}, workload)$

The function f is monotonically increasing with respect to the first and monotonically decreasing with respect to the second parameter. It can be observed that both these parameters are favourable in AFS compared to NFS. The temporal locality workload complements the block caching policy, resulting in over 90% cache hits [21]. Hence, AFS is more scalable compared to NFS. We have considered NFS versions less than 4 for the comparison. NFS 4 incorporates several features of AFS such as the use of callbacks for cache invalidation, delegation for file caching (same effect as whole file caching) etc.

5 Conclusions

We have presented a model for scalability in distributed systems. This model can be used to characterize the scalability of existing distributed systems, to compare similar systems with respect to scalability and to optimize existing systems in order to improve the scalability. The model can also serve as a guideline for designing new systems by considering related issues such as synchronization, consistency and availability. It complements existing scalability analysis models such as LQN based models. The scalability model opens up research in the distributed systems area (and related areas of distributed mobile systems and wireless adhoc networks) as it provides a unique characterization of scalability in terms of related factors such as synchronization, consistency and availability.

The scalability model can be applied to Wireless Sensor Networks (WSNs) to characterize scalability of WSNs. However, it is not clear if the current set of parameters considered is minimal and complete in this case. The issue of energy efficiency, so important in WSNs, does not arise in normal distributed systems. Another direction that we are taking is to proivde a solid theoretical foundation for the scalability model.

References

[1] Sanjay Ghemawat, Howard Gobioff, and Leung Leung. The Google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 96–108, Bolton Landing, NY, October 2003. ACM Press.

[2] Mahadev Satyanarayanan. The Influence of Scale on Distributed File System Design . *IEEE Transactions on Software Engineering*, 18(1):1–8, January 1992.

[3] Maarten van Steen and Philip Homburg and Andrew S. Tanenbaum. Globe: A Wide-Area Distributed System . *IEEE Concurrency*, 7(1):70–78, January-March 1999.

[4] Robbert Van Renesse, Kenneth P Birman and Werner Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003.

[5] Prasad Jogalekar and Murray Woodside. Evaluating the Scalability of Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*, 11(6):589–603, June 2000.

[6] J A Rolia and K C Sevcik. The Method of Layers. *IEEE Transactions on Software Engineering*, 21(8):689–700, August 1995.

[7] J E Neilson, C M Woodside, D C Petriu, and S Majumdar. Software Bottlenecking in Client-Server Systems and Rendezvous Networks. *IEEE Transactions on Software Engineering*, 21(9):776–782, September 1995.

[8] Douglas Schmidt, Michael Stal, Hans Robert, and Frank Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects.* John Wiley & Sons, Inc., 2000.

[9] Paul Brebner and Jeffrey Gosper. How scalable is J2EE technology? *ACM SIGSOFT Software Engineering Notes*, 28(3):4–10, May 2003.

[10] Seth Gibert and Nancy Lynch. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *ACM SIGACT News*, 33(2):51–59, June 2002.

[11] Haifeng Yu and Amin Vahdat. The Costs and Limits of Availability of Replicated Services. In *Proceedings of the ACM Symposium on Operating System Principl es (SOSP).* Banff, Canada, October 2001.

[12] Florin Sultan and Thu D. Nguyen and Liviu Iftode . Lazy Garbage Collection of Recovery State for Fault-Tolerant Distributed Shared Memory . *IEEE Transactions on Parallel and Distributed Systems*, 10(13):1085–1098, October 2002.

[13] Nicholas Carriero and David Gelenter. Linda in Context. *Communications of the ACM*, 4(32):444–458, April 1989.

[14] Michael Stumm and Songnian Zhou. Algorithms Implementing Distributed Shared Memory. *IEEE Computer*, 23(5):54–64, May 1990.

[15] A Vijay Srinivas, D Janakiram, and Raghevendra Koti. Virat: An Internet Scale Distributed Shared Object, Event and Service Space. Technical Report IITM-CSE-DOS-2004-03, Distributed & Object Systems Lab, Indian Institute of Technology, Madras, 2004.

[16] A Vijay Srinivas, Raghavendra Koti, A Uday Kumar, and D Janakiram. Realizing Large Scale Distributed Event Style Interactions. In *Proceedings of the European Conference on Object Oriented Programming (ECOOP) Workshop on Communication Abstractions for Distributed Systems.* Oslo, Norway, 2004.

[17] Chi Zhang and Zheng Zhang. Trading Replication Consistency for Performance and Availability: an Adaptive Approach. In *Proceedings of the 23rd International Conference on Distribut ed Computing Systems.* Providence, Rhode Island, USA, May 2003.

[18] Mustaque Ahamad and Rammohan Kordale. Scalable Consistency Protocols for Distributed Services. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):888–903, September 1999.

[19] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and Implementation of the Sun Network Filesystem. In *USENIX Association Summer Conference*, Portland, USA, 1985.

[20] Mahadev Satyanarayana. Influence of Scale on Distributed File System Design. *IEEE Transactions on Software Engineering*, 18(1):1–8, January 1992.

[21] Mirjana Spasojevic and M. Satyanarayana. An Empirical Study of a Wide-Area Distributed File System. *ACM Transactions on Computer Systems*, 2(14):200–222, May 1996.