Automating Service Quality with TOMCAD (Tradeoff Model with Capacity and Demand)

Raihan Al-Ekram, Ric Holt
University of Waterloo
{rekram,holt}@swag.uwaterloo.ca

Chris Hobbs
Nortel, Canada
cwlh@nortel.com

Susan Sim
University of California, Irvine
ses@ics.uci.edu

ABSTRACT

Large distributed networked software systems are built to provide competing qualities such as reliability, availability, security, performance and scalability to their clients. In certain situations these qualities must be traded off, sacrificing some qualities to some extent to improve others. This paper presents TOMCAD (a Tradeoff a Model with Capacity and Demand) for such tradeoffs and gives various properties and constraints that apply to such situations. With this approach, we show how to dynamically allocate system resources when demand changes with time, in a way that maintains required objectives for some service qualities. We demonstrate how this model applies to a family of distributed systems that trades off data consistency to gain availability.

Keywords

Tradeoff, QoS, TACT.

1. INTRODUCTION

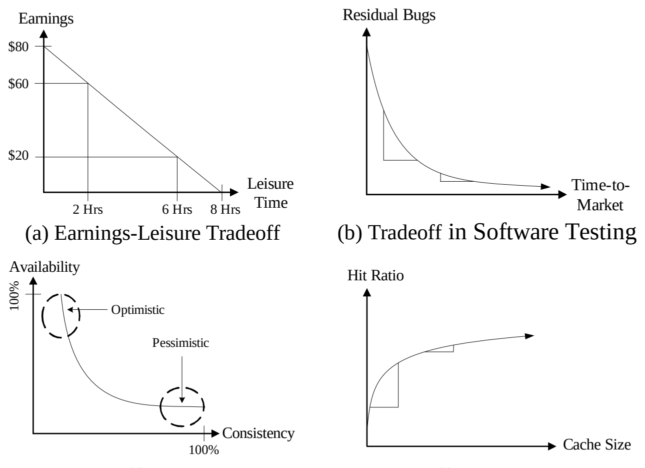
Large distributed networked software systems are built with competing qualities such as performance, scalability, reliability, availability and security to the user of their services. In general, all these qualities cannot be maximized at the same time. For example making remote procedure calls secure hampers their performance [4], increasing the performance of information retrieval on the web by caching comes at the cost of staleness [2] and providing high availability for mail or bulletin board services using lazy replication causes inconsistency among the replicas [5]. Tradeoff is inevitable between different quality attributes in such situations.

Tradeoff refers to a compromise between two or more different benefits. It happens in a situation when maximizing both the benefits at the same time is not possible to achieve. One of them can be sacrificed to some extent for gaining more of the other. A tradeoff is finding a right balance between both the benefits.

In this paper we establish a set of terminology and vocabulary to characterize tradeoffs. We discuss various concepts relevant to a system making some tradeoff and the relationship between them. We present a general model of tradeoff TOMCAD (a Tradeoff a Model with Capacity and Demand) in our effort to analyze and understand the nature of tradeoffs in software systems. The TOMCAD model has various properties and constraints that apply to a tradeoff situation. We also illustrate how to use the model to fulfill different QoS goals in a system with dynamically changing load and failure characteristics. We demonstrate the applicability of the model on a family of distributed systems built using the TACT (Tunable Availability and Consistency Tradeoffs) [10][11] or other TACT-like [7][12] middleware, that provides a systematic tradeoff between availability and data consistency of the service.

2. TRADEOFF EXAMPLES

This section introduces tradeoff in the area of economics, software engineering, distributed systems and computer architecture in terms of the examples illustrated in Figure 1.



(a) Earnings-Leisure Tradeoff (b) Tradeoff in Software Testing

(c) Tradeoff in Replication (d) Tradeoff in Memory Caching

Figure 1. Tradeoff Examples

2.1 Earnings-Leisure Tradeoff

There is a tradeoff between how much money someone can earn in a day and how much time he can spend in leisure. For a given amount of time a day, say 8 hours, he can spend all of the time working and earning money. Or he can spend some of it in leisure with his children. For fixed hourly pay, say $10/hr, every additional earning of $10 requires a sacrifice of an hour of leisure time. This tradeoff relationship between the dollars earned and the leisure time is depicted in the curve shown in Figure 1(a).

2.2 Tradeoff in Software Testing

Bugs are inevitable in a software system. A study [13] found an average of 0.33 bugs per KLOC in Linux Kernel 2.6, while Apache had an average of 0.25 bugs per KLOC. It is quite impossible to release a flawless software with many thousands of lines of code in it. The number of bugs in a software can be reduced by doing more testing before releasing it. Hence the managers in a software development project have to make a tradeoff decision between the time-to-market of the product and the acceptable residual bugs in the release. According to the software reliability growth models [6] more bugs are detected per unit time at the early stage of the testing period compared to the late stage, giving a software testing tradeoff curve as shown in Figure 1(b).

2.3 Tradeoff in Replication

Replication is a technique used to increase the reliability and availability of software systems. In a replicated system multiple copies of the software and hardware resources are maintained. The replication management protocol gives the illusion of a single system to its clients while the client requests are served by the replicas concurrently. In case of failures, the system can still continue to operate as long as there are some replicas operational, thus increasing the availability of the system. However replication introduces the issue of inconsistency among the replicas. The internal states of the replicas can diverge from each other as they process client requests concurrently. In order to limit the inconsistency among the replicas it is necessary to periodically synchronize them with each other.

There are two classes of replication techniques, namely pessimistic and optimistic. In *pessimistic* replication [3], any state changing operation in a replica (e.g., an update) is synchronously propagated to all other replicas maintaining a single copy consistency throughout the system. In a wide-area setting like the Internet, due to high network latency, frequent synchronization can cause increased client response time and lead to request timeouts in the clients. Also, due to link unreliability, access to the replicas may be denied when part of the network becomes temporarily unreachable. Hence pessimistic replication provides strong consistency but no guarantee on availability. On the other hand in *optimistic* replication [8], any kind of operation in a replica is allowed to proceed without any priori synchronization. Synchronization is done asynchronously in background. The time lag between an update in a replica and its propagation will cause stale reads in other replicas. Also, concurrent updates in different replicas may cause conflicts that have to be aborted eventually. Optimistic replication provides high availability but no guarantee on consistency.

Pessimistic and optimistic replications are the two extremes in the consistency and availability tradeoff as shown in Figure 1(c).

2.4 Tradeoff in Memory Caching

Cache memories are high-speed memories in between the CPU and main memory in a computer system to hold the portions of the main memory currently being used by the CPU. Cache memory is usually much faster than main memory but much smaller, as dictated by its cost. Memory pages requested by the CPU are loaded in the cache memory from the main memory, if it does not already exist in the cache. As new pages come in the cache old pages may have to be replaced according to criteria (e.g., LRU, LFU). The benefit of cache comes from the property of “locality of reference”, which means the address space to be used in near future is likely to be the current address space. The performance of cache memory is measured in terms of hit ratio, the probability of finding a memory request in the cache. Studies [9] have shown that the hit ratio increases with the cache size. Initially adding a small amount of cache increases the hit ratio significantly, but at a certain point the hit ratio becomes saturated and does not increase significantly with the addition of more cache. The curve in Figure 1(d) illustrates the tradeoff between cache hit ratio and cache size in a cache memory system.

3. TOM: THE BASIC TRADEOFF MODEL

In this section we present the basic tradeoff model TOM (Tradeoff Model) that gives the properties of tradeoff curves and the constraints for a tradeoff to exist in a system. TOM was described in our earlier work [1].

3.1 Morality: GG, BB, GB and BG

A (binary) tradeoff is a relationship between the benefits of two aspects of a system. However, the metric used to measure a property and to draw the tradeoff curve does not always indicate a benefit. Instead it may indicate the lack of the benefit. In the earnings-leisure tradeoff curve in Figure 1(a), more dollars earned along *Y*-axis and more leisure time along *X*-axis are both benefits. But In the software testing tradeoff example in Figure 1(b), more residual bugs along *Y*-axis and more time-to-market along *X*-axis are both measures for lack of benefits. The benefits are along the reverse directions of the axes, namely, less residual bugs and less time-to-market.

We define the morality of a tradeoff curve based on the goodness or badness of the properties along the axes of the curve. An axis has a **good** morality if it measures the goodness of a property, i.e., the benefit increases along the (positive) axis. On the other hand an axis has a **bad** morality if it measures the badness of a property, i.e., the benefit decreases along the axis. A tradeoff curve is said to have a **Good-Good (GG)** mortality if the benefits increase along both the *X* and *Y* axes of the curve. The earnings-leisure tradeoff curve of Figure 1(a) and the replication tradeoff curve of Figure 1(c) are examples of GG tradeoff curve. In these examples as *X* increases the benefits leisure time and consistency increase and as *Y* increases the benefits earnings and availability also increase.

In a **Bad-Bad (BB)** morality tradeoff curve the benefits decrease along both the axes. The software testing tradeoff curve of Figure 1(b) is a BB tradeoff curve, since the benefits lie along the reverse direction of both the axes. Similarly in a **Good-Bad (GB)** tradeoff curve the benefit increases along the *X*-axis, but decreases along the *Y*-axis. In a **Bad-Good (BG)** tradeoff curve the benefit decreases along the *X*-axis, but increases along the *Y*-axis. The memory cache tradeoff in Figure 1(d) is an example of BG tradeoff curve.

3.2 Monotonicity: Increasing or Decreasing

Whatever the morality of curve is, in order for it to be a tradeoff curve it has to fulfill the basic tradeoff condition: increase in one benefit will cause a decrease in the other benefit. As a result in a GG tradeoff curve as *X* increases to increase one benefit, *Y* necessarily decreases to decrease the other benefit. This constraint between *X* and *Y* dimensions of a GG tradeoff curve can be characterized as **Monotonically Decreasing (MD)**. In a BB tradeoff curve as *X* decreases to increase one benefit, *Y* will increase to decrease the other. This also gives a monotonically decreasing constraint. Conversely, in a GB tradeoff curve as *X* increases to increase a benefit, *Y* will increase to decrease the other benefit. A GB tradeoff curve can be characterized as **Monotonically Increasing (MI)**. Similarly in a BG tradeoff curve, as X decreases to increase a benefit, Y will also decrease to decrease the other benefit. This also gives a monotonically increasing curve.

3.3 Tradeoff Constraints

By combining the morality and the monotonicity properties we now define the basic constraint of tradeoff. For a tradeoff to exist between two GG or BB properties X and Y of a system, there must be a monotonically decreasing relationship between them. Eq. 1 expresses this constraint mathematically.

$$Y = f(X) \quad and \quad \frac{dY}{dX} < 0 \qquad \text{(Eq. 1)}$$

On the other hand the tradeoff constraint for two GB or BG properties X and Y of a system is for them to have a monotonically increasing relationship, which is given by Eq. 2.

$$Y = f(X) \quad and \quad \frac{dY}{dX} > 0 \qquad \text{(Eq. 2)}$$

Both X and Y are dependent on each other and their dependency is defined by Eq. 1 or Eq. 2. As such, the X and Y axes may not have any particular dependent or independent characterization. In other words changing value along any axis will cause the other to change accordingly.

An actual tradeoff curve may not be as smooth and ideal in shape as the curves in Figure 1. But there still can be a tradeoff as long as the appropriate constraint of Eq. 1 or Eq. 2 holds.

4. TOMCAD: TOM WITH CAPACITY AND DEMAND

In this section we extend TOM to TOMCAD, which combines the effect of capacity and demand of a system with the basic tradeoff described in the previous section. Let us revisit the software testing tradeoff example of Figure 1(b). The more testing is done on a software, the more bugs are found and resolved. More testing can be achieved by testing the software for a longer period. But the longer the testing period, the later the time-to-market for the product. The result is a tradeoff between the time-to-market of the software and the residual bugs. Since this is a tradeoff, it is not possible to reduce residual bugs and time-to-market at the same time. This tradeoff scenario has some underlying assumptions in it. Two important assumptions are that the system has a fixed capacity and demand. The capacity of a software testing system is how much code the team can test per unit time, which is determined by the size of the testing team. On the other hand the demand on a software testing system is the size of the code base to be tested, which is given by the number of features to be tested. It is possible to decrease both the residual bugs and the time-to-market of the product by either increasing the team size (the capacity) or decreasing the number of features in the software (the demand). The tradeoff given by the curve in Figure 1(b) assumes a particular capacity and demand of the system.

4.1 Family of Tradeoff Curves

We will illustrate the effects of capacity and demand on the software testing example with the curves in Figure 2. Suppose in the development of a particular software system there are M_1 different features to be tested by a team of size N_1. The tradeoff between residual bugs and time-to-market of the software is given by the curve R_1. For a target time-to-market x_1 of the product, the achievable residual bugs is y_1 given by the point p_1 on the curve. If this amount of residual bugs is not acceptable in the release and it has to be at most y_2, moving along curve R_1 up to point p_2 by increasing the time-to-market to x_2 can achieve this residual bugs target. But if it is required to meet the residual bugs target of y_2 without changing the time-to-market target of x_1, an alternative could be to increase the testing team size. For the new team size, for every time-to-market the amount of residual bugs will be lower than the previous team size. This will give a new tradeoff curve for this increased capacity and the new curve will be closer to the origin from its previous location. For some team size $N_2 > N_1$ both x_1 and y_2 target for time-to-market and residual bugs can be achieved at point p_3 on the new curve R_2. On the other hand decreasing the capacity of the testing team will move the tradeoff curve away from the origin. For some team size $N_3 < N_1$ the new tradeoff curve will be R_3. This gives a family of tradeoff curves based on the capacity of the system.

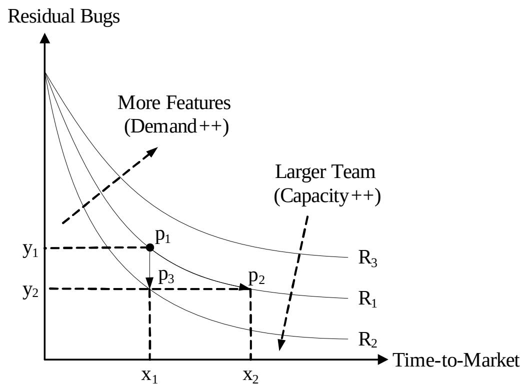


Figure 2. Capacity and Demand in Testing Tradeoff

The above analysis is done by varying the capacity and keeping the demand fixed. The same result can be achieved by keeping the capacity fixed and varying the demand for new features but in reverse direction. Increasing demand will move the tradeoff curve away from the origin, while decreasing demand will move the curve towards the origin. If curve R_1 represents a number of features M_1 and team size N_1, curve R_2 may represent a number features of $M_2 < M_1$ and R_3 may represent a number of features of $M_3 > M_1$ for the fixed team size N_1.

4.2 Capacity

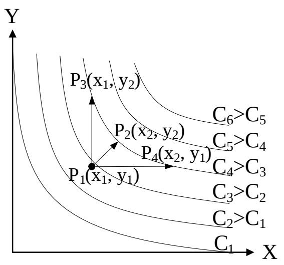
The capacity C of a system is the amount of resources available to perform its designated tasks. In a system making a tradeoff between two benefits X and Y, it is possible to gain more of both the benefits by increasing the capacity of the system. We assume X and Y to be in a GG tradeoff relationship in this discussion. The constraint among X, Y and C, where $C = f(X,Y)$, can be given as Eq. 3.

$$\frac{\partial C}{\partial X} > 0 \; and \; \frac{\partial C}{\partial Y} > 0 \qquad \text{(Eq. 3)}$$

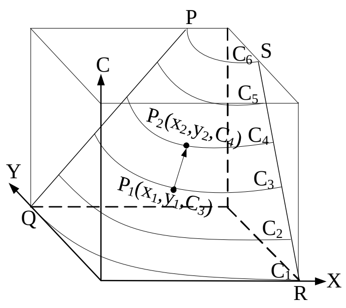
Since increasing C allows obtaining more of both X and Y, the new XY tradeoff curve will move away from the origin. Note that this is just the opposite from the software testing tradeoff example in the previous section, since the morality of the software testing tradeoff curve (it is BB) is just the opposite.

Notice that the relationship between C and either X or Y is monotonically increasing. The morality of the curve is GB since X or Y is a benefit (good) and more capacity indicates more cost (bad). But a curve with GB morality, which is also monotonically increasing, is the property of a tradeoff curve (Eq. 2). This results in a three-dimensional tradeoff among X, Y and C.

Figure 3(a) shows a family of XY tradeoff curves for various capacities of C, C_1 being the lowest capacity and C_6 being the highest. Suppose the system is operating at capacity C_3 and at tradeoff point P_1. By increasing capacity to C_4 the system moves to a new XY tradeoff curve. During the move from the original curve to the new curve, it is possible to increase Y without changing X at point P_3 or to increase X without changing Y at point P_4 or increase both X and Y by moving to point P_2.



(a) Capacity in 2D



(b) Capacity in 3D

Figure 3. Capacity in Tradeoff

The relationship among X, Y and C can be illustrated in the 3D graph of Figure 3(b), with C as the z-axis. This is the 3D equivalent of the 2D graphs in Figure 3(a). The tradeoff between X and Y as C varies is determined by the surface $PQRS$.

4.3 Demand

Some systems may have a variable amount of demand imposed on them. The demand on a system is the volume of task it has to perform. Increasing the demand on the system may reduce both of the basic benefits X and Y. The constraint among the demand on the system D, X and Y, where $D = f(X,Y)$, can be given as Eq. 4.

$$\frac{\partial D}{\partial X} < 0 \ \ and \ \ \frac{\partial D}{\partial Y} < 0 \qquad \text{(Eq. 4)}$$

As Eq. 4 specifies, increasing D reduces one or both of X and Y, so the new tradeoff curve will move towards the origin. The relationship among X, Y and D will have curves similar to Figure 3(a) and 3(b) except the direction of D will be reverse from that of C.

The relationship between D and either X or Y is monotonically decreasing. The morality of this curve is GG since X and Y are benefits (good) and more demand implies more revenue (good). A GG morality curve, which is also monotonically decreasing, is the property of a tradeoff curve (Eq. 1). This adds a fourth dimension D to the tradeoff among X, Y and C.

4.4 Capacity and Demand

Given fixed X and Y, the relationship between capacity C and demand D of the system is a two-dimensional tradeoff relationship. In order to handle more demand, without changing X and Y, the system needs more capacity. This is a monotonically increasing relationship between the capacity (bad) and demand (good). Eq. 5 gives the constraint between C and D of a system, where $C = f(D)$.

$$\frac{\partial C}{\partial D} > 0 \qquad \text{(Eq. 5)}$$

4.5 Four-way Tradeoff

Combining Eq. 1, 3, 4 and 5, the complete set of tradeoff constraints among GG properties X and Y, the capacity C and the demand D on the system is given by Eq. 6.

$$\begin{aligned} & f(X,Y,C,D) = 0 \quad where \\ & \frac{\partial Y}{\partial X} < 0,\ \frac{\partial C}{\partial X} > 0,\ \frac{\partial C}{\partial Y} > 0, \\ & \frac{\partial D}{\partial X} < 0,\ \frac{\partial D}{\partial Y} < 0 \ \ and \ \ \frac{\partial C}{\partial D} > 0 \end{aligned} \qquad \text{(Eq. 6)}$$

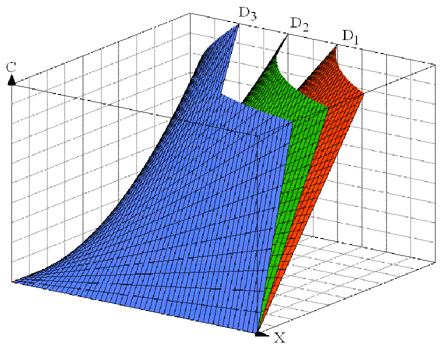


Figure 4. Four-way Tradeoff

Figure 4 illustrates the combined four-dimensional tradeoff relationship. The dimensions for each surface in the figure are X, Y and C. Each surface represents the tradeoff among X, Y and C for a particular D. The surfaces represent the demands D_1, D_2 and D_3 respectively, where $D_1 < D_2 < D_3$.

5. TOMCAD IN DISTRIBUTED SYSTEMS

Section 2.3 and Figure 1(c) describes the tradeoff between availability and consistency in a replicated system. Various middleware and frameworks are available to perform a continuous tradeoff between the two extremes of the figure. TACT (Tunable Availability and Consistency Tradeoffs) [10][11] is such a middleware toolkit for building replicated services. It provides a means to make a controlled and systematic tradeoff between the consistency and availability of the service by varying the level of inconsistency among the replicas. FRACS (Flexible Replication Architecture for a Consistency Spectrum) [12] is another TACT-like framework, which provides tradeoff between either performance or availability and consistency in replicated systems. The techniques in TRAPP (Tradeoff in Replication Precision and Performance) [7] enable tradeoff between performance and precision in data caching.

In this section we apply our tradeoff model on distributed systems built using TACT and analyze the resulting tradeoffs. From the analysis we show how to dynamically allocate system resources when demand changes with time, in a way that maintains a QoS guarantee to the user of the system.

5.1 TACT

In the TACT system model, as illustrated in Figure 5, the TACT middleware sits in between the application and the data store in each replica in order to manage the replication protocol. The replicas can directly communicate with each other. A client, intending to use the service, can submit a query to any of the replicas.

TACT controls consistency vs. availability tradeoff as follows. Each replica maintains measures of current inconsistency and maximum tolerable inconsistency between the data in it and that of other replicas in the system. For every client request to a replica (a read or a write operation), if performing the operation does not cause the current inconsistency to exceed the maximum tolerable inconsistency, the operation is done locally within the replica. Inconsistency is introduced in the system when a write operation is performed locally while letting the data in this replica diverge from the others. If performing an operation causes the current inconsistency to exceed the maximum tolerable inconsistency, the replica will first synchronize the accumulated write operations with one or more other replicas and reduce the current level of inconsistency in it.

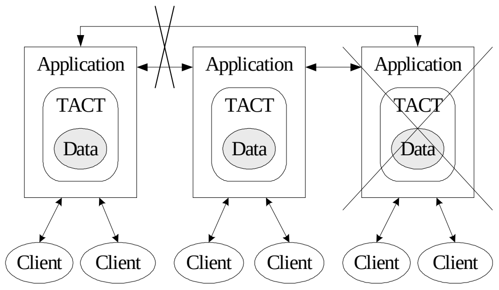


Figure 5. TACT System Model

It is inevitable that components of an Internet scale service will occasionally fail. The failure model of TACT handles two types of failure: replica failure and network partitioning. Any attempt to perform synchronization from a replica during a failure will be unsuccessful if it cannot communicate with the required replicas. This will cause the replica to reject any subsequent write operations from the clients, since it cannot do so without remaining in the tolerable inconsistency bound, until the component is repaired.

The higher the tolerable inconsistency bound, the higher the chances that a failed component will be repaired by the time a replica reaches the bound, hence the lower the number of rejected write requests and hence the higher the availability. This creates a tradeoff between the consistency and the availability of the system.

5.2 Analytical Model of TACT

We now build an analytical model of a TACT-based system consisting of the relationship among the relevant system parameters. From the model we then determine the tradeoffs in the system. Our model assumes Order Error [11] as the metric for measuring inconsistency. This metric specifies the number of write operations a replica can accumulate before synchronizing with the rest. In TACT the read operations are always handled locally, so they do not contribute to systems availability during a failure in other parts of the system. Hence only the writes are considered as client requests in our analysis. Our model has the following parameters for the system:

- *N* = Number of replicas, $N > 0$
- *E* = Order Error tolerated in each of the replicas, $1 \le E \le \infty$
- *D* = Total client demand in terms of number of writes per unit time arriving across the set of replicas, $D > 0$
- *L* = Total lost or rejected writes per unit time across all the replicas, $0 \le L \le D$
- *C* = Total capacity utilization to serve the demands across all replicas

For simplicity we assume that the lost operations (*L*) do not consume any system resources. So the effective total demand on the system is $D - L$ writes per unit time. This demand must be supplied by all *N* replicas, so the total capacity of the system needs is $N(D - L)$. Since we are using TACT, each replica accumulates its operations into batches of size *E*. These batches are transmitted to and effectively dealt with by the other replicas as a single operation. So the resulting required capacity is reduced by a factor of *E*. Thus the actual capacity utilization *C* is given by Eq. 7.

$$C = \frac{N}{E}(D - L) \qquad \text{(Eq. 7)}$$

We will now give an example of particular values that can satisfy Eq. 7. Consider a configuration in which the system has 3 replicas (*N*=3), the allowed order error in each replica is 5 (*E*=5), there is a total of 100 write operations per minute arriving at all the replicas (*D*=100), and a total of 5 of them are lost due to failures in the system (*L*=5). The resulting capacity utilization of the system is computed to be 57 write operations per minute (*C*=57). The combined capacity of all the replicas needs to be at least 57 write operations per minute. In the rest of the analysis we will assume that the capacity is exactly given by Eq. 7.

E and *L* are measures of badness, thus giving a BB tradeoff. In order to transform it into a GG tradeoff we can take the inverse of the measures. The inverse of *E* and *L* are the benefits consistency and availability. We will replace *E* and *L* with *X* and *Y* where

- *X* = Consistency of the system, $0 \le X \le 1$
- *Y* = Availability of the system, $0 \le Y \le 1$

When there is no order error in the system ($E = 1$), i.e., every operation is synchronized immediately, the system is fully consistent and $X = 1$. Whereas when order error is arbitrarily large ($E = \infty$), the system is totally inconsistent and $X = 0$. So *E* is transformed to *X*, for some constant $K > 0$, by Eq. 8.

$$X = \frac{K+1}{K+E} \quad \Rightarrow \quad E = \frac{K(1-X)+1}{X} \qquad \text{(Eq. 8)}$$

On the other hand the availability of the system, as defined in TACT, is the ratio between the number of fulfilled requests and the total requests. So *D* is transformed to *Y* by Eq. 9.

$$Y = \frac{D-L}{D} \quad \Rightarrow \quad L = D(1-Y) \qquad \text{(Eq. 9)}$$

Combining Eq. 7, 8 and 9 we get Eq. 10, a combined relationship among the system properties consistency *X*, availability *Y*, capacity *C*, demand *D* and number of replicas *N*.

$$C = \frac{NDXY}{K(1-X)+1} \qquad \text{(Eq. 10)}$$

5.3 Tradeoff Analysis

Based on the analytical model of TACT given in Eq. 10, we now derive the pair-wise monotonicity relationships among the GG properties *X* and *Y* along with the capacity *C* and demand *D* of the system. Eq. 11 lists the obtained relationships.

$$\frac{\partial Y}{\partial X} = -\frac{C(K+1)}{NDX^2} < 0 \qquad \frac{\partial^2 Y}{\partial X^2} = \frac{2C(K+1)}{NDX^3} > 0$$

$$\frac{\partial C}{\partial X} = \frac{NDY(K+1)}{[K(1-X)+1]^2} > 0 \qquad \frac{\partial C}{\partial Y} = \frac{NDX}{K(1-X)+1} > 0$$

$$\frac{\partial D}{\partial X} = -\frac{C(K+1\_}{NX^2Y} < 0 \qquad \frac{\partial D}{\partial Y} = -\frac{C[K(1-X)+1]}{NXY^2} < 0$$

$$\frac{\partial C}{\partial D} = \frac{NXY}{K(1-X)+1} > 0 \qquad \text{(Eq. 11)}$$

The set of equations of Eq. 11 satisfies the tradeoff constraints of TOMCAD given in Eq. 6. This confirms that there is a four-way tradeoff among the consistency, availability, capacity and demand of a service built using TACT.

5.4 Automating Service Quality

With the change in the system dynamics as the values of some of the system parameters change, the analytical model of Eq. 10 and the tradeoff relationships of Eq. 11 can be used to determine the values for the remaining parameters of the system. This model can also be used to determine how to adjust the parameters automatically to maintain a certain QoS guarantee to the user of the system, as we will now illustrate in the next paragraph.

Suppose that initially the system with capacity c_1 (*C*) is required to provide a particular minimum QoS specified as consistency x_1 (*X*) and availability y_1 (*Y*) while the total demand is d_1 (*D*). The actual initial values for *X* and *Y* are x_2 and y_2, such that $x_2 > x_1$ and $y_2 > y_1$. The QoS requirements for *X* and *Y* are clearly satisfied by the initial values. Now suppose that due to a system failure the capacity is reduced to c_2 such that $c_2 < c_1$. Consequently, the system can make a tradeoff between *C* and *X* and reduce *X* down to x_1 and/or make a tradeoff between *C* and *Y* and reduce *Y* down to y_1 to cope with the reduced capacity. If at this point the client demand increases to d_2 such that $d_2 > d_1$, this will try to push *X* and/or *Y* down. But since x_1 and y_1 are the minimum QoS required, the available options are either not to let the demand increase or to add more resources to the system to raise *C* to some amount c_3 such that $c_3 > c_2$ as required by the *C* and *D* tradeoff.

This approach leads to satisfying QoS by adjusting required parameters. The amount of adjustment required is given by the pair-wise tradeoff relationship between the relevant quality and the parameter.

6. CONCLUSION

In this paper we present TOMCAD, a tradeoff model that can be applied to various kinds of distributed systems in order to identify and understand the tradeoffs going on in the systems. TOMCAD implies that a tradeoff between any two qualities of a system is actually a four-way tradeoff. The capacity and the demand of the system introduce two additional dimensions to the basic tradeoff between system qualities, such as reliability, availability, security, performance and scalability. So in any tradeoff analysis it is important to take the additional two dimensions into account and consider how they might affect the tradeoff situation. We also show that by having a tradeoff model and the tradeoff equations, it is possible to automatically control system parameters and resource allocation to maintain a certain QoS goal for the service.

7. REFERENCES

[1] Raihan Al-Ekram, Ric Holt and Chris Hobbs, Applying a Tradeoff Model (TOM) to TACT. Proceedings of the 2nd International Conference on Availability, Reliability and Security, April 2007.

[2] Rafael Alonso, NJ Daniel Barbara and Hector Garcia-Molina, Data caching issues in an information retrieval system. ACM Transactions on Database Systems, Vol. 15(3), September 1990.

[3] Philip Bernstein and Nathan Goodman, The Failure and Recovery Problem for Replicated Databases. Proceedings of the 2nd Annual ACM symposium on Principles of Distributed Computing, 1983.

[4] Chi-Chao Chang, Grzegorz Czajkowski, Chris Hawblitzel, Deyu Hu and Thorsten von Eicken, Security versus performance tradeoffs in RPC implementations. Proceedings of the 8th ACM SIGOPS European Workshop on Support for composing distributed applications, September 1998.

[5] Rivka Ladin, Barbara Liskov, Liuba Shrira and Sanjay Ghemawat, Providing high availability using lazy replication. ACM Transactions on Computer Systems, Vol. 10(4), November 1992.

[6] J. D. Musa and A. F. Ackerman, Quantifying software validation: when to stop testing? IEEE Software, Vol. 6(3), May 1989.

[7] C. Olston and J. Widom, Offering a Precision-Performance Tradeoff for Aggregation Queries over Replicated Data. Proceedings of the 26th International Conference on Very Large Data Bases, September 2000.

[8] Yasushi Satio and Marc Shapiro, Optimistic Replication. ACM Computing Survey, Vol. 37(1), March 2005.

[9] Alan Jay Smith, Cache Memories. ACM Computing Surveys, Vol. 14(3), September 1982.

[10] Haifeng Yu and Amin Vahdat, Building Replicated Internet Services Using TACT: A Toolkit for Tunable Availability and Consistency Tradeoffs. Proceedings of the 2nd Workshop on Advanced Issues of E-Commerce and WeBBased Information Systems, June 2000.

[11] Haifeng Yu and Amin Vahdat, Design and Evaluation of a Conit-Based Continuous Consistency Model for Replicated Services. ACM Transactions on Computer Systems, Vol. 20(3), August 2002.

[12] Chi Zhang and Zheng Zhang, Trading Replication Consistency for Performance and Availability: an Adaptive Approach. Proceedings of the 23rd International Conference on Distributed Computing Systems, 2003.

[13] Joab Jackson, Open-source Bug Hunt Results. http://www.gcn.com/online/vol1_no1/40053-1.html