Minimal Replication Cost for Availability [*]

Haifeng Yu
Computer Science Department
Duke University
Durham, NC 27708
yhf@cs.duke.edu

Amin Vahdat
Computer Science Department
Duke University
Durham, NC 27708
vahdat@cs.duke.edu

ABSTRACT

Today, the utility of many replicated Internet services is limited by availability rather than raw performance. To better understand the effects of replica placement on availability, we propose the problem of *minimal replication cost for availability*. Let replication cost be the cost associated with replica deployment, dynamic replica creation and teardown at n candidate locations. Given client access patterns, replica failure patterns, network partition patterns, a required consistency level and a target level of availability, the minimal replication cost is the lower bound on a system's replication cost. Solving this problem also answers the dual question of optimal availability given a constraint on replication cost.

We design the first algorithm we are aware of to solve the problem, through reduction to integer linear programming and enumeration of pruned serialization orders. Using practical faultloads and workloads, we demonstrate that the exponential complexity of our algorithm is tractable for practical problems with hundreds of candidate locations. The lower bound computed by our algorithm is tight, but the tightness can be sacrificed by a proposed optimization for large problems. We also show that with low replica creation and teardown costs, the bound is close to tight in practical problems even with the optimization.

1. INTRODUCTION

Replication has long been a key approach to achieving high availability [5, 17, 23]. Today, wide-area replication is further facilitated by web hosting services and content delivery networks. A key issue in this environment is the placement of replicas across a set of candidate locations. With changing network conditions and client access patterns, a system may also need to decide when replicas should be dynamically created or torn down (i.e., dynamic replica placement). Typically, replication systems strive to minimize replication cost—the cost associated with replica deployment, creation and teardown—while reaching a performance or availability target (or to maximize performance and availability given a constraint on replication cost).

Replica placement and replication cost have been widely studied for performance [14, 16, 21, 25]. However, there is much less research on replica placement for availability, especially in WAN settings. Compared to optimal performance, studying replica placement for availability requires a fundamentally different model to accommodate dynamic network partition scenarios. Traditionally, availability studies have been restricted to the database context [3, 9]. Placement has not been a primary concern in these settings because the studies typically focus on static, small-scale configurations. Given that the utility of many current network services is limited by availability rather than raw performance [11] (e.g., even a 1% improvement in availability can provide 3.6 additional days of service uptime per year), we believe that the problem of dynamic replica placement for availability (or simultaneous performance and availability targets) is important. To the best of our knowledge, our study is the first to address the replica placement problem for optimal availability in partitionable networks.

The interaction between availability and consistency makes the problem of replica placement for availability challenging. Link failures in the Internet can result in complex network partition scenarios. For example, in our experiments we observe up to 18 coexisting partitions. In a partitioned network, a particular consistency level (e.g., one-copy serializability [4]) may restrict the set of acceptable accesses even if all requests reach at least one replica. Such restrictions rule out the trivial solution of placing one replica in each partition, given the goal of minimizing overall replication cost. Such effects become more significant if the availability target is not 100% and the system needs to decide which partitions to sacrifice. The set of acceptable accesses in one partition depends on the consistency level, accesses being accepted in other partitions and the history of system-wide accepted accesses. Computing the globally optimal placement thus becomes a complex combinatorial optimization problem. Furthermore, to study the lower bound on replication cost, we cannot assume any specific consistency protocols (e.g., quorum protocols [3]) and we must study the

[*]This research is supported in part by the National Science Foundation (EIA-9972879, ITR-0082912), Hewlett-Packard, IBM, Intel, and Microsoft. Vahdat is also supported by an NSF CAREER award (CCR-9984328).

inherent impact of consistency on availability.

We now provide a more detailed description of the *minimal replication cost (for availability)* problem. Define replication cost to be the sum of the cost of replica creation, replica teardown and replica usage. The network has n candidate locations for replica deployment and m clients accessing the service. A workload describes the time-stamped accesses from all m clients. Both replicas and links in the network may fail, which are fully described by a *faultload.* The system may specify a required consistency level among replicas using the TACT continuous consistency model [27]. Many discrete consistency levels, including one-copy serializability [4], linearizability [12] and delta consistency [24], can be expressed as instances of continuous consistency. Availability is defined to be the number of accepted accesses over total submitted accesses in the workload. Given a workload, faultload, consistency level and an availability target, the *minimal replication cost (for availability)* is the lower bound on replication cost of any system that can reach the availability target. Solving this problem essentially gives us minimal replication cost as a function of an availability target, and thus also answers the following question: Given a constraint on replication cost, what is the optimal availability?[1] The end goal of this work is to allow researchers to rigorously compare the costs of real replication systems against optimal bounds and to provide insights into how existing systems can be optimized for availability.

In this paper, we formalize the minimal replication cost problem for availability and design an algorithm to solve it. Given a faultload, we reduce the problem to an integer linear programming (IP) problem based on an *evolution graph.* However, the reduction is dependent on the system's serialization order and the number of serialization orders grows exponentially with the number of accepted writes. To make the search tractable, we define a *domination* relationship among serialization orders and aggressively prune non-optimal serialization orders. We also prove that logical serialization orders can be used to represent physical ones and thus avoid enumerating all possible sets of accepted writes. These techniques, together with our optimizations in Section 6, reduce the number of serialization orders enumerated from $10^{200,000}$ to well below $5,000$ in our target faultloads. Finally, we obtain the minimal replication cost by enumerating the resulting small set of logical serialization orders, solving an IP for each order and finally choosing the minimal solution.

Our algorithm is the first that we are aware of to solve the minimal replication cost problem for availability. The algorithm can be combined with the replica placement for performance [21] to solve the problem with dual performance and availability targets (details available in the full version [28] of this paper). The lower bound computed by our algorithm is tight, but the tightness can be sacrificed if we use an important optimization for large problems. Using faultloads based on Internet-like topologies [29] and workloads based on web client traces [8, 19], we show that with low replica creation and teardown costs, the bound is close to tight even with all our optimizations. While our algorithm is exponential ([28] proves that the problem is NP-hard), we demonstrate that the complexity is tractable for practical problems with hundreds of candidate locations and day-long workloads and faultloads.

[1]We do not use this alternative form of the problem so that we can add a dual performance target.

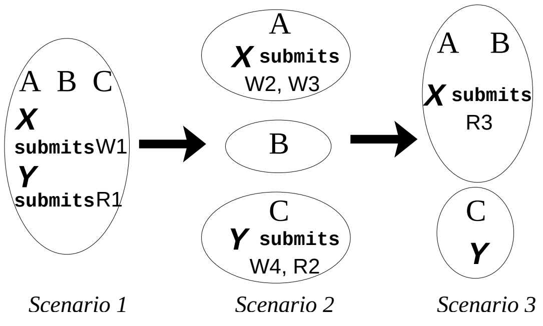

Figure 1: An example with three candidate locations (A, B and C) for placing replicas and two clients (X and Y). W_1, W_2, W_3 and W_4 are writes submitted by clients, while R_1, R_2 and R_3 are reads.

The next section gives a simple example and describes related work. We formalize the minimal replication cost problem in Section 3. Section 4 solves the special case of the problem under one-copy serializability [4] (1SR) and linearizability [12] (LIN). We then generalize to continuous consistency in Section 5. Section 6 discusses two important optimizations and Section 7 presents our experience based on practical faultloads and workloads. Finally, Section 8 presents our conclusions.

2. BACKGROUND

2.1 Motivating Example

To further illustrate the minimal replication cost problem and the challenges in solving it, in Figure 1 we give a concrete example with three candidate locations and two clients. For simplicity, we only consider network partitions with no replica failures.[2] The system goes through three partition scenarios, and we annotate what accesses are submitted by the clients during each scenario. If no consistency is required, then the minimal replication cost (assuming we only consider usage cost) can be determined trivially. In this example, the minimal cost for 100% availability is achieved when we place a replica at A throughout the execution and a replica at C during the second partition scenario.

However, the previous placement becomes suboptimal if we enforce LIN for the execution. To ensure LIN, during the second partition scenario, only one of the two replicas at A and C will be able to accept accesses. Furthermore, if we allow the replica at C to accept W_4 and R_2, then R_3 cannot be accepted during *Scenario 3* by a potential replica at either A or B, because the replica will not see W_4 before processing R_3. Since some of the replicas cannot accept accesses anyway, in order to achieve minimal cost, they should not be deployed. However, whether a replica can accept an access depends on the previous execution of the system. For

[2]In fact, Section 4 will show that replica failures can be modeled as network partitions.

example, we can have the replica at C reject W_4 while still accepting R_2, to allow the future acceptance of R_3. This will help increase availability if R_2 and R_3 both represent many reads instead of a single read. Thus, we see that consistency requirements restrict the set of accesses that can be accepted and the acceptable sets of accesses by each partition are inter-dependent. Continuous consistency further increases the challenge by allowing a bounded amount of inconsistency.

2.2 Related Work

Under the assumption of 1SR, Coan et.al. [9] derive tight availability upper bounds for two-way partition scenarios without replication cost constraints, avoiding much of the complexity with complex partition scenarios. Johnson et.al. [13] prove that for update transactions and a single data item, replication provides little availability benefits relative to an optimally placed centralized server under 1SR. Replica placement studies in Farsite [6, 10] and RMS [18] target replication on local-area networks and thus assume no network partitions.

Recently, several studies solve the replica placement problem for optimal performance [14, 16, 21, 25]. These studies address different versions of the problem by imposing various constraints (e.g., number of replicas [16, 21], storage space [14] and bandwidth [25]) on the replication system. More specifically, Li et.al. [16] study optimal replica placement in tree topologies. The assumption on topology allows the authors to develop a polynomial algorithm using dynamic programming. Optimal replica placement in arbitrary topologies can be trivially reduced to a facility location problem as in [21]. Similar to our approach, Qiu et.al. [21] use IP to solve the problem and then mainly concentrate on comparing the optimal placement against various placement heuristics. However, because their work does not consider dynamic network partition scenarios or consistency, the reduction to IP is trivial. For example, there is no need for evolution graphs or considering different serialization orders. The studies on space-constrained [14] and bandwidth-constrained [25] replica placement are based on the assumption of a simplified hierarchical distance model. However, the studies also impose constraints on the capacity of replicas, resulting in a different challenge. By reducing the placement problem to min-cost flow, polynomial placement algorithms are designed for both problems. There is no straightforward way to extend their results to availability, since dynamic network partition scenarios cannot be easily accommodated in the min-cost flow problem. Finally, consistency is not considered in their studies, which greatly simplifies the problem.

In this work, we use the consistency model developed in our earlier efforts [27], which defines the TACT continuous consistency model and argues for its generality and practicality. The domination definition among serialization orders in Section 5.1 is adapted from a slightly different version in our previous work [26] on real replication system availability. This earlier work assumes reads are always accepted and discusses the best availability for writes given a fixed replica set. By assuming reads are always accepted, the study excludes protocols that can disable one partition to allow progress in other partitions. Furthermore, in [26] we concentrate on comparing the achieved availability of various consistency protocols against the best availability and show how the protocols can be optimized. On the other hand, this work focuses on the minimal replication cost for dynamic replica placement given n candidate locations, and determines which replicas should be disable to achieve optimal availability.

3. PROBLEM FORMULATION

3.1 Failure Model and Availability Definition

For simplicity, we refer to application data as a database, though the data can actually be stored in a database, file system, persistent objects, etc. The database is replicated in full at multiple *replicas*. Replicas may be dynamically created or torn down and we assume that a replica loses its state when torn down. However, if a replica crashes and then recovers, it is assumed to recover its state for stable storage. To prevent permanent data loss, we assume at least one replica in the system at all times. Each replica may accept *reads* and *writes* from clients, both called *accesses*. Our reads and writes are query transactions and update transactions in database terminology. For simplicity, we assume the database has only one data item.

Both replicas and the network may experience fail-stop failures. We assume that network failures are symmetric. Without loss of generality, we also assume reachability among machines is transitive for the purpose of calculating lower bound. With the symmetry and transitivity assumptions, network failures break the network into *partitions*. A *faultload*, which is a trace of timestamped *failure events* and *recovery events* for replicas and the network, fully specifies the failure pattern. For example, a failure event may describe how the network is partitioned. We do not explicitly discuss network topology, since it is already abstracted in the faultload.

We define *Availability = accepted accesses / submitted accesses*. A *submitted access* from a client is an *accepted access* if the client can reach some replica that can accept the access. Notice a functioning replica may not always be able to accept an access due to consistency requirements. For instance, assuming a quorum protocol, replicas unable to collect the necessary quorum must reject an access. A *workload* fully describes the time-stamped accesses submitted from clients, i.e., when and which client submits which access.

Our faultload approach significantly differs from traditional approaches of studying availability [10, 13] where failures are probabilistically modeled instead of deterministically described. We apply this approach because, unlike replica failures, there is currently no convincing way to mathematically model the occurrences of network partitions. It is also NP-hard [22] to derive the partition model from link and node failure models. With the faultload approach, the lower bound on cost depends on faultloads instead of failure models. One advantage of our approach is that the lower bound theory is independent of the faultloads, while studies based on specific probabilistic failure models are usually difficult to extend to other failure models.

3.2 Replication Cost Model and Problem Definition

The most straightforward definition for replication cost is the number of replicas. However, this definition cannot distinguish how long a replica is needed at a location under dynamic replica placement. Thus, we use the following general definition as the *replication cost* of an execution. For each candidate location i, the cost definition takes into account the usage cost($\alpha \times usage_i$), creation cost($\beta \times create_i$) and teardown cost($\gamma \times teardown_i$) of replica incarnations at that location:

$$Cost = \sum_{i=1}^{n}(\alpha \times usage_i + \beta \times create_i + \gamma \times teardown_i)$$

In the equation, $usage_i$ is the total time that $location_i$ has a replica and α is the cost per unit time. The variable $create_i$ denotes the number of replica creations at $location_i$ and β is the creation cost. Similarly, $teardown_i$ and γ represent replica teardown costs. With different α, β and γ values, the cost function can emulate different metrics. For example, with α and γ being zero, the cost function becomes the number of replicas. The values of α, β and γ can also vary from location to location in a straightforward manner.

We can now formally define the *minimal replication cost* problem: Given 1) n candidate locations for placing replicas; 2) a workload; 3) a faultload; 4) desired consistency level (1SR, LIN or continuous consistency [27]); and 5) an availability target to achieve, what is the lower bound on replication cost?

4. MINIMAL REPLICATION COST UNDER 1SR AND LIN

As discussed earlier, the main challenge in the minimal replication cost problem is the interaction between availability and consistency. Generally speaking, the higher the consistency level, the lower the availability, and thus the higher the replication cost to achieve a given availability target. Traditionally, system designers are forced to make a binary decision between two extremes of consistency, 1SR/LIN or no consistency at all, each with its own associated tradeoffs. A number of efforts [15, 20, 27] argue for the benefits of a *continuous consistency model*, where consistency is quantified and dynamically adjustable. For example, in a traffic monitoring system with continuous consistency, the application can specify the maximum allowed staleness of the traffic data. In a distributed game, users may specify the maximum allowed error in the observed position of various objects in the virtual world to trade accuracy for performance and availability. A full discussion of the benefits and applicability of continuous consistency is beyond the scope of this paper. For our study, a continuous model allows a more complete exploration of the problem space. As special cases, our theory still applies to traditional consistency models such as 1SR and LIN.

We use the TACT continuous consistency model [27] in this study because of its generality. We have shown [27] that many previous consistency models, including 1SR, LIN, conflict matrix [2], N-ignorant system [15], delta consistency [24] and quasi-copy caching [1], can be expressed as instances of the model. We have also demonstrated how a wide range of applications (e.g., bulletin boards and airline reservations) can utilize the model.

For clarity, we discuss the simple cases of 1SR and LIN in this section and extend to continuous consistency in the next section.

4.1 Effects of Dynamic Replica Placement

Our replication model allows replicas to be dynamically created and torn down. To depict the possible behavior of a location, we divide the execution into *intervals*, which are periods of time during which network connectivity does not change. The faultload determines the maximal length of the intervals, but intervals can be infinitely short. We assume that replicas can only be created or torn down at the boundary of intervals, so we can discretize dynamic replica deployment. Using short intervals (e.g., on the order of seconds) will help reduce the error introduced by this discretization process. We will quantify this error for practical scenarios in Section 7.

We use a boolean variable $dep_{k,i}$ to denote whether a replica is deployed at $location_i$ during $interval_k$. For simplicity, we assume that replica creation is instantaneous. Replica creation delay (i.e., the delay before a replica can accept accesses after its creation) can easily be accomodated by properly adjusting replica creation cost. For example, if the time needed to create a replica is 5 minutes, we can increase replica creation cost by 5 minute*replica. It can be easily proven that solving the problem with the new replica creation cost assuming instantaneous replica creation is equivelant to solving the orginial problem with non-zero replica creation delay.

At each interval, the accesses in one partition may be accepted only if there is at least one replica in that partition. Notice that we need not separately address replica failures because they are equivalent to one-node partitions (under the assumption that failed replicas can recover their states from stable storage). Let $writes_{k,m}$ be the number of writes accepted by $partition_m$ during $interval_k$, and let $wsubmit_{k,m}$ be the number of writes submitted from clients. Let $deptn_{k,m} = \vee\{dep_{k,i}|location_i \in partition_m$ during $interval_k\}$, which denotes whether $partition_m$ has at least one replica deployed during $interval_k$. To avoid excessive notation, we use boolean variables as binary variables in arithmetic operations and we have: $writes_{k,m} \leq deptn_{k,m} \times wsubmit_{k,m}$. Similar constraint can be constructed for reads. These constraints and the objective (replication cost) for the IP problem can all be expressed linearly (see [28]). The only constraints missing here are consistency constraints, which will be discussed next. In fact, solving such an IP problem without consistency constraints results in the minimal replication cost when no consistency is required (or for read-only services).

4.2 Effects of 1SR and LIN

1SR requires the execution of the replication system to be equivalent to a serial execution on a single database. With 1SR, a replica can always accept reads. However, during any interval, only one partition (the *primary partition*) can accept writes (remember we assume a single data item in

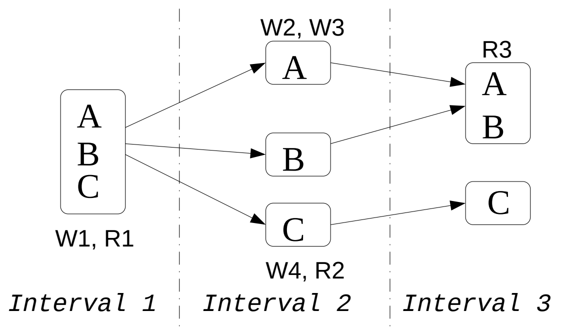

Figure 2: The evolution graph of Figure 1.

the database). Furthermore, two consecutive primary partitions must intersect, so that the writes accepted by the primary in one interval can be transmitted to the primary in the subsequent interval. We now formalize and prove the restrictions imposed by 1SR[3].

Definition 1. *The* **evolution graph** *of a faultload is a directed graph constructed as follows: For each interval in the faultload, add a node to the graph for each partition in that interval. Let* $node_{k,m}$ *correspond to* $interval_k$, $partition_m$. *An edge from* $node_{k,m}$ *to* $node_{k',m'}$ *is added if* $k' = k+1$ *and* $partition_{m'}$ *intersects with* $partition_m$ *at one or more candidate locations. A node in the evolution graph is an* **ancestor** *of another node if there is a path from the former to the latter.*

Figure 2 illustrates the evolution graph of the execution in Figure 1. For clarity, we do not plot client machines. Each box in the figure represents a partition, and beside each box we annotate those accesses submitted within the partition.

Lemma 1. *Given any execution that satisfies 1SR, let S be the set of all (interval, partition) tuples where the partition accepts at least one write during the interval. Then S is covered by a path in the evolution graph, i.e., all nodes in S belong to a particular path.*

Proof: See [28]. □

Based on this lemma, we only need to enumerate all possible paths in the evolution graph to determine which partitions can accept writes. Notice we actually only enumerate those longest paths, i.e., the paths whose length equals to the total number of intervals. For each path, we dictate that the nodes not covered by the path cannot accept any writes. Adding these further constraints to our IP problem results in a minimal cost for each possible path. The minimal cost across all possible paths will be the minimal replication cost.

LIN is strictly stronger than 1SR in that it requires the execution to be equivalent to a serial execution on a single database where the accesses are ordered in the same order as they are accepted by the system. A system under 1SR can always accept reads because reads can be "serialized" to the beginning of the execution. With LIN, a read needs to observe all writes that are accepted before the read. In Figure 2, C's acceptance of W_4 prevents A from accepting R_3 because after W_4 is accepted, A becomes "inconsistent" and cannot process any access until it sees W_4. With the same discretization technique as before, we use $able_{k,i}$ to denote whether $location_i$ is consistent ("able" to accept accesses) during $interval_k$. Formally, suppose $node_{k,m}$ is the node in the evolution graph containing $location_i$ in $interval_k$. Let $unseen_{k,i}$ be the set of writes accepted by all nodes $node_{k',m'}$ such that i) $k' \leq k$; ii) $node_{k',m'} \neq node_{k,m}$ and iii) $node_{k',m'}$ is not an ancestor of $node_{k,m}$. LIN dictates that if $able_{k,i} = 1$, $unseen_{k,i}$ must be empty. We can express this requirement in linear form using a constant MAX that is large enough (e.g., MAX can be the total number of writes in the workload): $|unseen_{k,i}| \leq (1 - able_{k,i}) \times MAX$.

The variable $able_{k,i}$ has a similar effect on availability as $dept_{k,i}$. Let $abletn_{k,m} = \vee\{able_{k,i} | location_i \in partition_m\ interval_k\}$ and we have: $writes_{k,m} \leq abletn_{k,m} \times wsubmit_{k,m}$. A similar constraint on reads can be constructed. As for 1SR, we solve an IP problem for each path enumerated from the evolution graph. The full version of this paper [28] proves the correctness and tightness of the lower bound.

5. MINIMAL REPLICATION COST UNDER CONTINUOUS CONSISTENCY

In this section, we extend our previous results to continuous consistency. For brevity, we will not rigorously define and justify the various metrics in the TACT continuous consistency model [27]. Rather, we focus on the model's effects on minimal replication cost. In TACT, consistency is quantified using three per-replica metrics: *order error*, *numerical error* and *staleness*. By setting different bounds on the three metrics, the model achieves different semantics. For example, zero order error is equivalent to 1SR, while requiring zero numerical error and zero order error corresponds to LIN.

5.1 Effects of Order Error

Order error can be viewed as a relaxation from 1SR. Recall that 1SR allows at most one primary partition in each interval in order to maintain a total order (*serialization order*) among all accepted writes. Non-zero order error bounds allow the system to relax this total order requirement. More specifically, order error is the number of writes that are out of order (according to the serialization order) at each replica. Consider the following example at the end of the second interval in Figure 2. Suppose the system has accepted four writes, W_1, W_2, W_3 and W_4 and the replica at location A has seen W_1, W_2 and W_3. With serialization order $W_1W_4W_2W_3$, the replica has two out of order writes, W_2 and W_3, since it does not see W_4 and the two writes are ordered after W_4 in the serialization order. Order error can be decreased by filling in "holes" in the serialization order. If A sees W_4 later on, all writes will become in order.

A non-zero order error bound allows non-primary partitions to accept writes, and Lemma 1 no longer holds. We now

[3]Such formalization based on evolution graphs may not be necessary for the simple case of 1SR, but we still use such formalization to prepare for continuous consistency.

need to explicitly check the order error at each replica. Formally, suppose the node containing $location_i$ of $interval_k$ in the evolution graph is $node_{k,m}$. Let $seen_{k,i}$ be the set of writes accepted by $node_{k,m}$ and all its ancestors. In other words, we partition all writes accepted by the system by the end of $interval_k$ into two disjoint sets $unseen_{k,i}$ and $seen_{k,i}$, for any valid i. Let $applied_{k,i}$ be the set of writes that a potential replica at $location_i$ applies to its local database at the end of $interval_k$, which can be any subset of $seen_{k,i}$. Let $inorder_{k,i}$ be the set of writes in $seen_{k,i}$ that are in order (according to a given serialization order). The replica's order error then equals $|applied_{k,i} - inorder_{k,i}|$ and it must be within specified bound $bound_{OE}$ if the replica is considered consistent[4]:

$$applied_{k,i} \subseteq seen_{k,i} \qquad (1)$$
$$|applied_{k,i} - inorder_{k,i}| \leq bound_{OE} + (1 - able_{k,i}) \times MAX \qquad (2)$$

In the inequalities, $inorder_{k,i}$ depends on the serialization order. Enumerating all possible serialization orders is impractical, given that they can be any total order of accepted writes. Furthermore, we do not yet know the exact set of accepted writes. In the following, we will gradually distill "better" serialization orders, with the ultimately goal of reducing the set to a small size for practical problems. We further show it is possible to enumerate those orders without even knowing which writes are accepted.

We first fix the set of accepted writes and define the "better" relationship among serialization orders. Directly defining "better" serialization orders that result in lower replication cost can be confusing. Instead, we define that with "better" serialization orders, more writes will be "in order":

DEFINITION 2. *Given an evolution graph and the set of accepted writes, serialization orders D_1* **dominates** *serialization order D_2 if the $inorder_{k,i}$ based on D_1 is a super set of the $inorder_{k,i}$ based on D_2, for any valid k and i.*

It is easy to see that this definition does result in "better" serialization orders in terms of replication cost, since any execution that uses D_2 can also use D_1 without violating any of the constraints (see [28] for formal proof).

Our first step of distilling serialization orders is based on the *ancestor* partial order among writes. A write W_1 is an *ancestor* of another write W_2 if either i) the node in the evolution graph accepting W_1 is an ancestor of the node accepting W_2 or ii) W_1 and W_2 are accepted by the same node and W_1 is accepted before W_2. Intuitively, since the ancestors of a write are always seen before the write itself, it is "better" to place the ancestors before the write in the serialization order. We construct $ANCESTOR$, which is the set of serialization orders that are compatible with the ancestor partial order. In Figure 2, a serialization order for W_1, W_2, W_3 and W_4 can be any of the 24 possible permutations of the four writes, while $ANCESTOR$ only contains $W_1W_2W_3W_4$, $W_1W_2W_4W_3$ and $W_1W_4W_2W_3$. The following lemma sketches why $ANCESTOR$ dominates all serialization orders, with full proof in [28].

[4]Setting $applied_{k,i}$ to $\emptyset$ will trivially satisfy the constraints, but the constraint on numerical error in the next section rules out this trivial solution.

LEMMA 2. *For any serialization order D, there exists another serialization order $D' \in ANCESTOR$ such that D' dominates D.*

Proof sketch: We re-arrange D step by step until it becomes compatible with ancestor order, while ensuring that the new serialization order dominates D. Consider any two writes W_1 and W_2, where W_1 is an ancestor of W_2. Suppose W_2 precedes W_1 in $D = S_1W_2S_2W_1S_3$, where S_1, S_2 and S_3 are arbitrary write sequences. We construct $D' = S_1W_1W_2S_2S_3$. It is easy to see that D' dominates D because W_1 is an ancestor of W_2 and any location that sees W_2 must have seen W_1.

Next, careful global re-arrangements can be done according to the topological ordering of the writes, so that each step does not introduce new conflicts with ancestor order. Finally, we have a serialization order that dominates D and that is compatible with the ancestor order. □

For our next step of distillation, define $CLUSTER$ to be the set of serialization orders that are in $ANCESTOR$ and where writes accepted by the same node in the evolution graph cluster together. In our previous example with four writes, $CLUSTER$ only contains $W_1W_2W_3W_4$ and $W_1W_4W_2W_3$, since W_2 and W_3 are accepted by the same node and thus cluster together. The intuition of this step is that for writes accepted by the same node, either all or none of them belong to $seen_{k,i}$ (for any k and i), thus it never hurts to cluster them together. The following lemma shows that $CLUSTER$ dominates $ANCESTOR$, with full proof in [28].

LEMMA 3. *For any serialization order $D \in ANCESTOR$, there exists another serialization order $D' \in CLUSTER$ such that D' dominates D.*

Proof sketch: We will again adjust D step by step until it is in $CLUSTER$, while ensuring the new serialization order dominates D. Without loss of generality, suppose $D = S_1W_1S_2W_2 \ldots S_nW_nS_{n+1}$, where W_1, W_2, ..., and W_n are the writes accepted by the same node in the evolution graph. We construct $D' = S1W_1W_2 \ldots W_nS_2S_3 \ldots S_nS_{n+1}$. Because D is compatible with ancestor order, it can be shown that no writes in S_2, S_3, ..., S_n are ancestors of W_1, W_2, ..., or W_n. Thus D' is compatible with ancestor order also. To prove that D' dominates D, notice our adjustment "pushes back" the sequences S_2, S_3, ..., S_n toward the end of the serialization order. If a write W, $W \in \{S_2 \cup S_3 \cup \ldots \cup S_n\}$, is in order based on D, the location must have seen W_1. Since W_1, W_2, ..., W_n are all accepted by the same node in the evolution graph, we know the location must have seen all of them. It is then easy to see W is in order based on D'.

We have shown that D' dominates D. Apply the previous re-arrangement for each node in the evolution graph and we will obtain a serialization order in $CLUSTER$ that dominates D. □

Hence, a small set of serialization orders, $CLUSTER$, dominates all serialization orders. However, the set $CLUSTER$ depends on the set of accepted writes. To avoid considering different sets of accepted writes, we use the concept of *logical writes* and *logical serialization orders* (a sequence of logical writes). We let each node $node_{k,m}$ in the evolution graph accept a logical write that stands for $writes_{k,m}$ number of physical writes. A location's order error is computed to be the total number of physical writes in out-of-order logical writes. If all logical writes stand for one or more physical writes, this is equivalent to using physical serialization orders. However, since some of the logical writes may stand for zero physical writes, false "holes" may appear in the sequence. Fortunately, we can show that this does not affect correctness:

LEMMA 4. *Given a logical serialization order $D = Y_1Y_2 \ldots Y_n \in CLUSTER$, suppose logical serialization order $D_1 = X_1X_2 \ldots X_m$ is obtained by deleting those logical writes in $Y_1Y_2 \ldots Y_n$ that stand for zero physical writes. Then there exists another logical serialization order $D_2 \in CLUSTER$, such that D_2 dominates[5] D_1.*

Proof sketch: We construct $D_2 = Z_1X_1Z_2X_2 \ldots Z_mX_mZ_{m+1}$ from D by "pushing" all logical writes in D that stand for zero writes as far back as possible without violating ancestor order. It then follows that $D_2 \in CLUSTER$ since we carefully preserve ancestor order. Next, we show that for any physical write in X_i, if it is considered in order based on D_1, it must be considered in order based on D_2. This is true because the logical writes in Z_i are all ancestors of X_i for $1 \leq i \leq m$, and if a location sees $X_1X_2 \ldots X_j$, it must have seen $Z_1X_1Z_2X_2 \ldots Z_jX_j$ for $1 \leq j \leq m$. □

With the previous lemma, we know it is safe to use logical serialization orders in place of physical ones and the set $CLUSTER$ is actually the set of all topological orderings of the evolution graph as a DAG.

5.2 Effects of Numerical Error and Staleness

Numerical error can be viewed as a relaxation from LIN. Informally, numerical error is the number of "missing" writes on a replica. LIN requires numerical error to be zero for all consistent replicas. With continuous consistency and a numerical error bound of, for example, 2, if C accepts W_4 in Figure 2, A may still accept R_3 because its numerical error is only one.

Formally, using our previous notations, the numerical error of the replica at $location_i$ at the end of $interval_k$ is $|unseen_{k,i}| + |seen_{k,i}| - |applied_{k,i}|$. Suppose $bound_{NE}$ is the maximal allowed numerical error, we have:

$$|unseen_{k,i}| + |seen_{k,i}| - |applied_{k,i}| \leq bound_{NE} + (1 - able_{k,i}) \times MAX \qquad (3)$$

[5]Here we slightly abuse the definition of "dominate", since D_1 and D_2 contain different sets of logical writes. However, the meaning should be clear from the context.

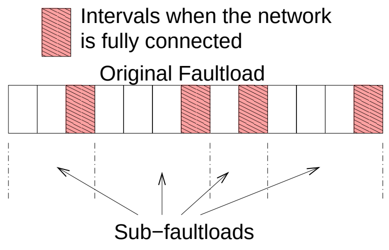

Figure 3: Splitting a faultload into sub-faultloads.

The third metric in the consistency model, staleness, bounds the age of missing writes. For any write W accepted at wall clock time t, a replica must see W by time $t + bound_{ST}$ in order to be consistent, where $bound_{ST}$ is the bound on staleness. For any $interval_k$, define $endtime(interval_k)$ to be the wall clock time that $interval_k$ ends. Then a consistent replica in $interval_k$ must see all writes accepted before time $endtime(interval_k) - bound_{ST}$. Without loss of generality, we assume $endtime(interval_k) - bound_{ST}$ falls on the boundary of two intervals: We can always split the interval if it falls in the middle of an interval. Define $stale_k$ to be the set of all writes accepted during $interval_{k'}$ such that $endtime(interval_k) - endtime(interval_{k'}) \geq bound_{ST}$. Given that for $location_i$ in $interval_k$, the set of missing writes are $unseen_{k,i} \cup (seen_{k,i} - applied_{k,i})$, staleness imposes the following constraint:

$$|(unseen_{k,i} \cup (seen_{k,i} - applied_{k,i})) \cap stale_k| \leq (1 - able_{k,i}) \times MAX \qquad (4)$$

Inequality 1, 2, 3 and 4 are the new inequalities for the IP problem under continuous consistency. Finally, we compute the minimal replication cost under continuous consistency by enumerating logical serialization orders based on the evolution graph. We construct an IP problem for each order enumerated, and choose the smallest cost across all serialization orders as the lower bound. The full version of this paper [28] proves the correctness and tightness of the lower bound.

6. OPTIMIZATIONS

Because we enumerate serialization orders and use IP in our algorithm, the worst case complexity of the algorithm is exponential. In this section, we present two important optimizations to make the computation tractable for relatively large problems. First, we observe that many candidate locations (up to 90% of all candidate locations depending on the faultload) are always connected with each other throughout practical faultloads. Intuitively, these locations are "equivalent" and we can delete all but one, thus reducing the number of binary variables in the IP problem by up to 10 fold. Since IP solvers have exponential complexity, this reduction is significant. The formal correctness proof for this optimization is available in [28].

The second optimization we apply is to calculate local optimal values instead of a global optimal value, by "splitting" a long faultload into many sub-faultloads. Our algorithm implicitly assumes that at the beginning of the execution,

Faultload	Failure Inter-Arrival Mean for Node	Failure Duration Mean for Node	Failure Inter-Arrival Mean for Link	Failure Duration Mean for Link
FL100:0.1%	5 days	1 minute	20 weeks	1 minute
FL100:0.5%	1 day	1 minute	4 weeks	1 minute
FL100:1%	12 hours	1 minute	2 weeks	1 minute
FL100:3%	12 hours	3 minutes	2 weeks	3 minutes
FL600:1%	12 hours	1 minute	2 weeks	1 minute

Table 1: Parameters used in generating our faultloads.

the replicas are fully "in sync." To make sure such assumption is satisfied at the beginning of each sub-faultload, we can only split the faultload at intervals when the network is fully connected (Figure 3). In order words, all sub-faultloads (except the last one) should end with a fully-connected interval. During any fully-connected interval, inconsistency can be fully resolved and thus the replicas are fully "in sync" again. In our experiments, we split one-day long faultloads into 20 to 50 sub-faultloads, which helps to reduce the size of $CLUSTER$ by up to 10^{20} fold in large problems. Because replication cost is a summation, the sum of local minimal costs is a lower bound on global replication cost. However, because we do not include replica creation/teardown cost across sub-faultloads, the bound is no longer tight. To see how close to tight the bound is, we add the worst-case creation/teardown cost across the sub-faultloads to the lower bound, and obtain an *achievable replication cost*. The closer it is to the lower bound, the closer to tight the lower bound is. In the next section, we will discuss how close to tight the bound is in practical problems.

7. EXPERIMENTAL RESULTS

We implemented our algorithm and present the results in this section. The main purpose of this section is to validate the practicality of our algorithm and we defer the evaluation of replication cost under various configurations to the full version [28] of this paper. This section: i) shows that the exponential complexity of our algorithm is tractable in practical cases; ii) demonstrates that the error introduced by discretization is small; iii) shows that the minimal replication cost is close to tight under low replica creation and teardown costs.

We first describe our experimental configuration. Because there are no Internet traces that measure the connectivity matrix among hundreds of nodes, we obtain our faultloads using an event-driven simulator based on Internet-like topologies [29]. The sample Internet topologies only contain routers. For each router in the topology, we attach two end machines, one as a candidate location for replicas, and another as a client. We then inject failure and recovery events to the nodes and links in the topology and faultloads are generated by computing the connected components of the graph as a function of time. Both failure inter-arrival time and duration are exponentially distributed. We use two different topologies, one with 100 candidate locations, and the other has 600 candidate locations. We vary the parameters in the failure model and obtain a series of faultloads: "FL100:0.1%", "FL100:0.5%", "FL100:1%", "FL100:3%" and "FL600:1%". For example, the faultload "FL100:0.5%" is one based on 100 candidate locations, with an average path failure rate of 0.5%. The average path failure rate describes how often two nodes in the network cannot communicate on average. We center the failure rate around 1% because one recent study [7] shows a 1.5-2% average failure rate in today's Internet. Table 1 summarizes the parameters we use for the exponential distributions. For node failures, we also use a correlation model. Whenever a node fails, we assume that the entire domain where the node resides completely fails (because of correlated failures) with 1% probability and that its adjacent nodes fail with 5% probability. All our faultloads are one-day long. For workloads, we use two publicly available web client traces [8, 19] and a synthetic one based on a Poisson distribution of request arrivals. We vary required consistency level from LIN to no consistency and the availability target from 99% to 100%.

The computational complexity of our algorithm is largely determined by the number of serialization orders enumerated and the size of the IP problems. Both factors are affected by the number of intervals in the sub-faultloads, which in turn depends on how often the candidate locations become fully connected with each other. Because the complexity of the algorithm is exponential, the time to solve the problem for a faultload is often dominated by the time to solve for the longest sub-faultload. For our faultloads, the length of the longest sub-faultload varies from 10 to 60 intervals (roughly 10 to 60 minutes in realtime). Given a sub-faultload, the size of the IP problem is determined by the number of candidate locations after applying the optimization in Section 6. In our experiments, the optimization can reduce the number of candidate locations from hundreds to tens, since many of them are connected with each other throughout a single sub-faultload. With a fixed sub-faultload, the number of enumerated serialization orders is mainly determined by the number of partitions containing writes and their ancestor relations. For partitions without submitted writes, we do not need to introduce logical writes for them. In our sub-faultloads, the total number of partitions varies from 20 to 844, among which 7 to 98 partitions have writes. At any single point of time, there are usually only a small number of partitions coexist, with the maximum number ranging from 4 to 18 depending on the faultload. The full version of this paper [28] provides full characterization of each faultload based on these metrics.

For all faultloads with 100 candidate locations, our algorithm enumerates less than 5,000 serialization orders and the total number of variables in each IP problem is on the order of thousands. These figures are consistent with the number of intervals, the number of candidate locations and

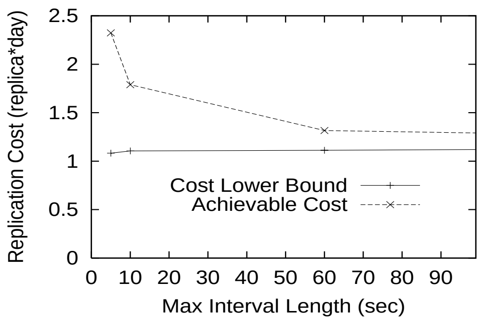

Figure 4: Minimal replication cost as a function of interval length.

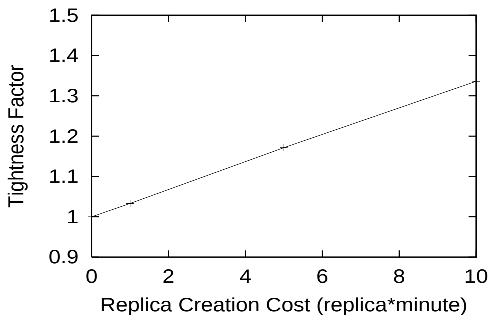

Figure 5: Tightness factor as a function of replica creation cost.

the number of partitions described earlier. The computation time ranges from 10 minutes to 12 hours on a Sun Ultra-5 workstation. The variation on computation time across experiments is large, mainly because the IP solving time is highly dependent on the parameters. For "FL600:1%", the problem can be solved within one day with the NASA [19] workload. With other workloads, the algorithm does not finish after three days' computation. From our experiments, we believe our algorithm's complexity is tractable for practical problems with hundreds of candidate locations. Many further optimizations can be applied to solve larger problems (e.g., use approximate IP solvers).

In our algorithm, we use discrete variables $dep_{k,i}$ and $able_{k,i}$ for each interval and location to approximate a continuous process. With this discretization technique, the shorter the interval length, the more accurate the results. Figure 4 plots the minimal replication cost and achievable replication cost as a function of the maximal length of an interval under "FL100:1%", the ClarkNet [8] workload, LIN and 99.7% availability target (highest availability achievable under this configuration). The lower bound curve is roughly flat, meaning the error introduced by discretization is small. Results for other configurations are similar. The figure also shows that with larger interval length and fewer sub-faultloads, the bound becomes tighter. Based on these results, we set the maximum interval length to 60 seconds in all our other experiments.

The original tightness of our lower bound is broken by the sub-faultload optimization in Section 6 and we now study how close to tight the bound is with the optimization. Let *tightness factor* be the ratio between achievable replication cost and the replication cost lower bound. The closer the tightness factor is to 1.0, the closer to tight the bound is. Our first set of experiments use 5 replica*minute creation cost and zero teardown cost, which means the cost for creating a replica is equivalent to the cost of using it for 5 minutes. For all configurations, the tightness factor is consistently below 1.20. We also experiment with different creation and teardown costs, and observe a roughly linear relationship between creation/teardown cost and tightness factor. Figure 5 presents the results under "FL100:1%", the ClarkNet [8] workload, no consistency and 100% availability target. Results on other configurations are similar. The tightness of the bound can be improved by using longer sub-faultloads, which also increases computation complexity. The one-hour long sub-faultloads we use provide some intuition behind the results: The minimal replication cost for each sub-faultload is on the order of one hour*replica, while the extra creation and teardown cost at the sub-faultload boundary is on the order of 5 replica*minute. In summary, we believe the bound is fairly close to tight under low replica creation and teardown cost.

8. CONCLUSIONS

Motivated by replicated Internet services, web hosting services and content delivery networks, we propose and formalize the problem of minimal replication cost for a given availability target. Through pruned serialization order enumeration and integer linear programming, we design the first algorithm we are aware of to solve this problem. The lower bound can help guide future design of replication systems. Based on Internet-like topologies and web client traces, we demonstrate that the exponential complexity of the algorithm is tractable for practical problems with hundreds of candidate locations. We also show that the bound is close to tight in practical problems with low replica creation and teardown cost.

9. ACKNOWLEDGMENTS

We would like to thank the anonymous reviews for their insightful feedback.

10. REFERENCES

[1] Rafael Alonso, Daniel Barbara, and Hector Garcia-Molina. Data Caching Issues in an Information Retrieval System. *ACM Transactions on Database Systems*, September 1990.

[2] B. R. Badrinath and K. Ramamritham. Semantics-based Concurrency Control: Beyond Commutativity. *ACM Transactions on Database Systems*, March 1992.

[3] Daniel Barbara and Hector Garcia-Molina. The Vulnerability of Vote Assignments. *ACM Transactions on Computer Systems*, August 1986.

[4] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[5] Andrew D. Birrell, Roy Levin, Roger M. Needham, and Michael D. Schroeder. Grapevine: An Exercise in Distributed Computing. *Communications of the ACM*, 25(4):260–274, April 1982.

[6] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2000.

[7] Bharat Chandra, Mike Dahlin, Lei Gao, and Amol Nayate. End-to-End WAN Service Availability. In *Proceedings of the 3rd Usenix Symposium on Internet Technologies and Systems*, January 2001.

[8] ClarkNet Server Traces. `http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html`.

[9] Brian Coan, Brian Oki, and Elliot Kolodner. Limitations on Database Availability When Networks Partition. In *Proceedings of the 5th ACM Symposium on Principle of Distributed Computing*, pages 187–194, August 1986.

[10] John R. Douceur and Roger P. Wattenhofer. Competitive Hill-Climbing Strategies for Replica Placement in a Distributed File System. In *Proceedings of the 15th International Symposium on Distributed Computing (DISC)*, pages 48–62, October 2001.

[11] John Hennessy. The Future of Systems Research. *IEEE Computer*, 32(8):27–33, August 1999.

[12] Maurice Herlihy and Jeannette Wing. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Transactions on Programming Languages and Systems*, 12(3), July 1990.

[13] Donald B. Johnson and Larry Raab. A Tight Upper Bound on the Benefits of Replication and Consistency Control Protocols. In *Proceedings of the 10th ACM Symposium on Principles of Database Systems*, May 1991.

[14] Madhukar Korupolu, Greg Plaxton, and Rajmohan Rajaraman. Placement Algorithms for Hierarchical Cooperative Caching. In *Proceedings of the 10th Annual Symposium on Discrete Algorithms*, January 1999.

[15] Narayanan Krishnakumar and Arthur Bernstein. Bounded Ignorance: A Technique for Increasing Concurrency in a Replicated System. *ACM Transactions on Database Systems*, 19(4), December 1994.

[16] B. Li, M. Golin, G. Italiano, and A. K. Sohraby. On The Optimal Placement of Web Proxies in the Internet. In *Proceedings of IEEE INFOCOM'99*, March 1999.

[17] K. Marzullo and F. Schmuck. Supplying high availability with a standard network file system. In *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS)*, 1988.

[18] Daniel L. McCue and Mark C. Little. Computing Replica Placement in Distributed Systems. In *Proceedings of the Workshop on the Management of Replicated Data*, 1992.

[19] NASA Kennedy Space Center Server Traces. `http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html`.

[20] Calton Pu and Avraham Leff. Replication Control in Distributed System: An Asynchronous Approach. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, May 1991.

[21] Lili Qiu, Venkata Padmanabahn, and Geoffrey Voelker. On the Placement of Web Server Replicas. In *Proceedings of the INFOCOM 2001 - Twentieth Annual Joint Conference of the IEEE Computer And Communications Societies*, April 2001.

[22] A. Rosenthal. Computing the Reliability of a Complex Network. *SIAM Journal of Applied Mathematics*, 32:384–393, 1977.

[23] J.B. Rothnie and N. Goodman. A Survey of Research adn Development in Distributed Database Systems. In *Proceedings of the 3rd International Conference on Very Large Databases*, October 1977.

[24] Aman Singla, Umakishore Ramachandran, and Jessica Hodgins. Temporal Notions of Synchronization and Consistency in Beehive. In *Proceedings of the 9th ACM Symposium on Parallel Algorithms and Architectures*, June 1997.

[25] A. Venkataramani, P. Weidmann, and M. Dahlin. Bandwidth Constrained Placement in a WAN. In *Proceedings of the Twentieth ACM Symposium on the Principles of Distributed Computing (PODC2001)*, August 2001.

[26] Haifeng Yu and Amin Vahdat. The Costs and Limits of Availability for Replicated Services. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.

[27] Haifeng Yu and Amin Vahdat. Design and Evaluation of a Conit-based Continuous Consistency Model. *ACM Transactions on Computer Systems*, August 2002.

[28] Haifeng Yu and Amin Vahdat. Minimal Replication Cost for Availability. Technical Report CS-2002-04, Duke University, Computer Science Department, Durham, NC, 2002. Available at `http://www.cs.duke.edu/~yhf/podc02tr.pdf`.

[29] Ellen W. Zegura, Kenneth Calvert, and M. Jeff Donahoo. A Quantitative Comparison of Graph-Based Models for Internet Topology. *IEEE/ACM Transactions on Networking*, 5(6), December 1997.