A Generalized Replica Placement Strategy to Optimize Latency in a Wide Area Distributed Storage System

John A. Chandy
Department of Electrical and Computer Engineering
University of Connecticut

International Workshop on Data-Aware Distributed Computing
June 24, 2008

Distributed Storage

- Local area network
  - Spread data across multiple networked nodes
  - Parallelism and higher throughput
- Wide-area network
  - Instead of splitting data for scalability, replicate data for availability
  - Improves latency

Replica Placement

- Where do you put the replicas?
  - Optimization problem
    - Minimize latency or maximize availability
    - Constraints: storage capacity, load balancing
  - Significant existing work
    - Latency optimization
      - Greedy algorithm, Qiu et al.
      - HotZone, Szymaniak et al.
        - » Popularity based
      - Lat-cdn, Pallis et al.
        - » Heuristic approach

Replica Placement

- Availability optimization
  - Van Renesse
    - Place replicas until desired availability is reached
  - Farsite
    - Hill-climbing approach to replica placement
  - Xin et al.
    - Takes into account bimodal availability

Replica Placement

- What's the problem?
  - Existing approaches assume that objects are completely replicated
  - Full replication has significant overhead
  - Use erasure codes instead
    - Less overhead
    - Better reliability than parity
    - Placement is much more complicated

Problem Formulation

- K data objects
- N storage nodes
- C clients
- Each object is split into n fragments of which m fragments must be recovered to reconstruct object
  - $m=n$ - no redundancy
  - $m=n$-1 - parity
  - $m=1$ - replication

Problem Formulation

- Placement problem
  - Place fragments of each object on n of the N storage nodes
  - $x_{jk}=1$ if fragment of object j is placed on storage node k
- Assignment problem
  - For each object, assign each client to m of the n storage nodes where the object fragments are placed
  - $y_{ijk}=1$ if client i retrives fragment of object j from storage node k

Problem formulation

- Overall objective is to minimize average latency

$$\begin{pmatrix} 0 & \lambda_{1,2} & \cdots & \lambda_{1,N-1} & \lambda_{1,N} \\ \lambda_{2,1} & 0 & & \lambda_{2,N-1} & \lambda_{2,N} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \lambda_{C-1,1} & \lambda_{C-1,2} & & 0 & \lambda_{C-1,N} \\ \lambda_{C,1} & \lambda_{C,2} & \cdots & \lambda_{C,N-1} & 0 \end{pmatrix}$$

- Cost function is

$$F(X,Y) = \sum_{k \in K} \sum_{j \in N} \sum_{i \in C} y_{ijk} \lambda_{ij}$$

Problem formulation

- Constraints:
  - x is a binary variable

    $$x \in \{0,1\}$$

  - y is a binary variable

    $$x \in \{0,1\}$$

  - Each object k has n fragments

    $$\sum_{j \in N} x_{jk} = n \quad \forall k$$

  - Each client i requests m fragments of object k

    $$\sum_{j \in N} y_{ijk} = m \quad \forall i,k$$

Problem formulation

- Constraints:
  - Client i should request a fragment of object k from storage node j only if that node stores the fragment

$$y_{ijk} \le x_{jk} \quad \forall i, j, k$$

  - Storage allocation balancing - each node j stores the same number of fragments

$$\sum_{k \in K} x_{jk} = \frac{nK}{N} \quad \forall j$$

  - Load balancing - each node j services the same number of clients

$$\sum_{i \in C} \sum_{k \in K} y_{ijk} = \frac{mCK}{N} \quad \forall j$$

Problem formulation

- 0-1 integer linear programming problem
  - $CN + CNK$ variables
  - $K + CK + CNK + N + N$ constraints
  - K and C can be in the millions and N can be in the thousands
  - Problem is too large to solve with normal methods

Problem formulation

- Instead of doing global placement, do a placement on each individual object
  - Makes more sense since it is impractical to reallocate and replace fragments every time an obect is created
  - Make x and y independent of k
    - Will cause load imbalance as all objects will be placed on the same nodes
  - Intermediate approach - consider only a subset of objects
    - Since each object has n fragments, we can insure that each node has at least 1 fragment, by setting $K = \frac{N}{n}$ and introducing new constraint $\sum_{k \in K} x_{jk} = 1 \;\; \forall j$

Problem formulation

- With reduced object set size, derive global placement P
- For each new object, calculate a hash h based on object ID, name, contents, etc.
- Place and assign object according to object $h \bmod K$ in placement P.

Problem Approach

- Heuristic approach to 0-1 integer linear programming problem
- Start with an initial placement and assignment that is guaranteed to be feasible
  - Place first object on the first n nodes, place second object on the next n nodes, and so on

$$x_{jk} = \left(\left\lfloor \frac{j}{n} \right\rfloor == k \right)$$

  - Assign first client to the first m nodes of the n storage nodes, next client to the next m nodes and so on

$$y_{ijk} = \left(\left\lfloor \frac{j}{m} \right\rfloor == k \frac{n}{m} + i \bmod \frac{n}{m} \right)$$

Problem Approach

- Greedily alter solution until no improvements
- 3 possible solution transformations
  - Swap assignment

    $y_{ijk} \leftrightarrow y_{i'j'k'}$ where $y_{ijk} = y_{i'j'k'} = 1$

  - Swap placement

    $x_{jk} \leftrightarrow x_{j'k'}$ where $x_{jk} = x_{j'k'} = 1$

  - Change assignment

    $y_{ijk} \leftrightarrow y_{i'j'k'}$ where $y_{ijk} = 1$ and $y_{i'j'k'} = 0$

Problem approach

- Change assignment can introduce load imbalance
- We can relax the load balance requirement

$$\frac{mCK}{N}(1-\lambda) < \sum_{i \in C} \sum_{k \in K} y_{ijk} < \frac{mCK}{N}(1+\lambda) \quad \forall j$$

Algorithm

```
cost = F( X, Y )
do
   oldcost = cost
   for all objects k
      for all clients i
         for all storage nodes j
            delta_cost = change assignment
            if ( delta_cost < 0 )
                accept change
   for clients with maximum latencies
      delta_cost = swap placement
      if ( delta_cost < 0 )
         accept swap
      delta_cost = swap assignment
      if ( delta_cost < 0 )
         accept swap
   cost = F(X,Y)
while cost < oldcost
```

Algorithm

- Swaps provide the most improvement, but are very costly to evaluate
- Assignment changes provide relatively little improve, but is very easy to evaluate
- Do mostly assignment changes and do swaps only for maximum latency clients
- $O(CN)$ computation due to $mCK=mCN/n$ non-zero elements in matrix

Evaluation

- Autonomous System Network generated with Inet topology generator
- Similar to a content delivery network with storage nodes at AS nodes
- Graphs with 3200, 4000, 5000, and 6000 nodes
- Clients and storage nodes are equivalent
- Latency is number of hops

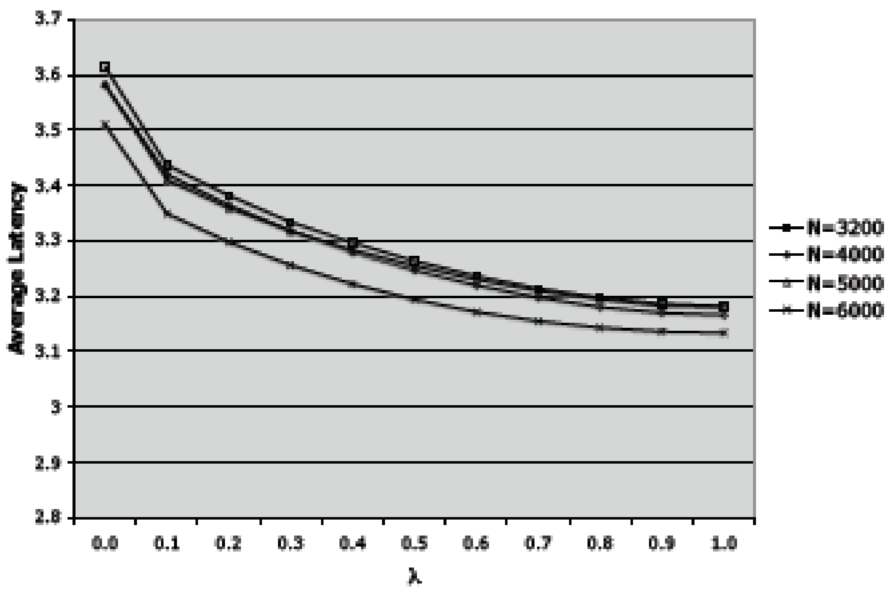
Evaluation

- N=4000, n=8, m=4

	Initial	Random	$\lambda = 0$	$\lambda = 0.2$	$\lambda = 1.0$
Average latency	3.582	3.156	3.581	3.365	3.166
Max latency	9	7	9	9	7
Std. Deviation	0.838	0.715	0.837	0.844	0.726
Average load	2000	2000	2000	2000	2000
Max load	2000	20461	2000	2400	4000
Min load	2000	0	2000	1600	4
Std. Deviation	0	2589	0	349	1065

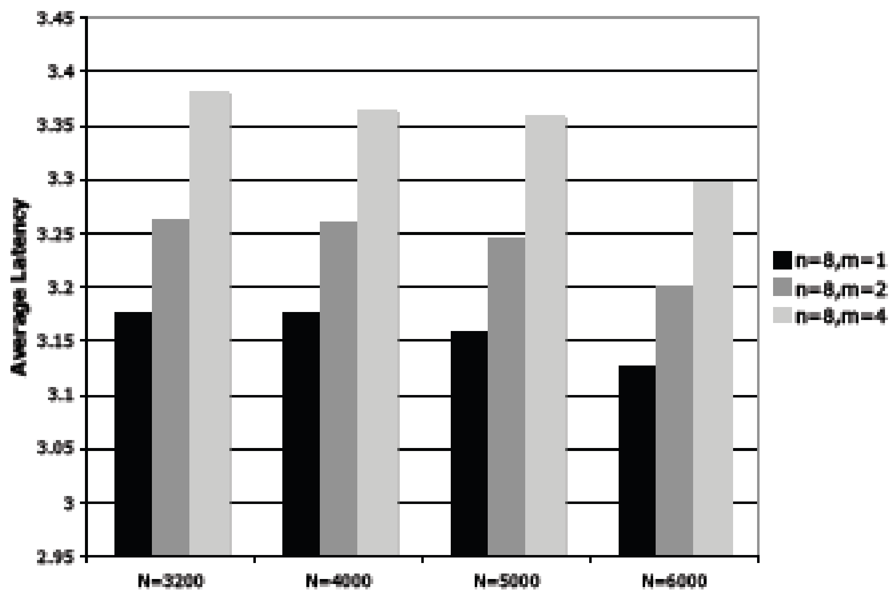
Evaluation

- Average latency vs. λ (n=8, m=4)



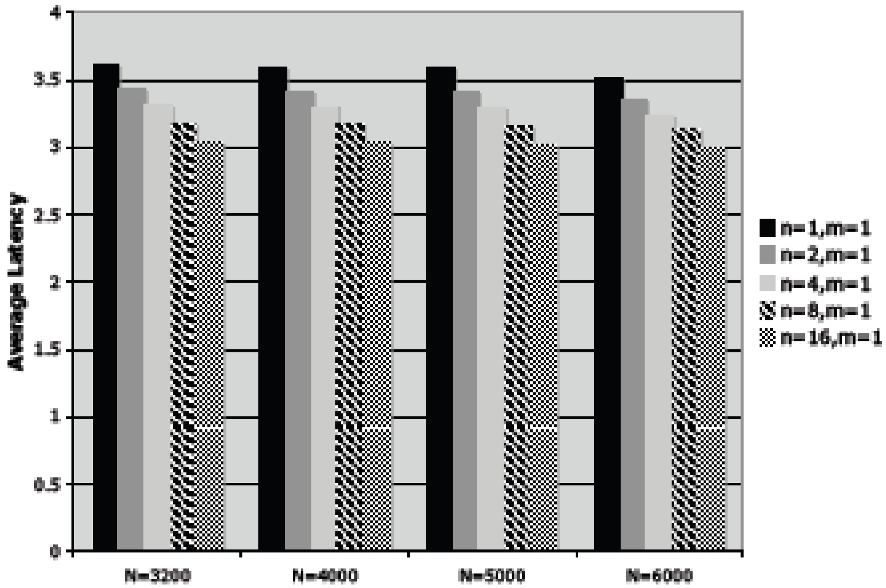
Evaluation

- Average latency vs. N (varying n,m)



Evaluation

- Average latency vs. N (varying n,m=1)



Summary

- Generalized replica placement algorithm suitable for fragmented objects - parity, erasure codes, secret sharing, etc.
- Greedy algorithm based on assignment changes and swaps
- Load balancing relaxation improves performance significantly

University of Connecticut