Partial Replica Selection for Spatial Datasets

Yun Tian (Tony), Philip J. Rhodes
Department of Computer and Information Science
University of Mississippi
Scientific datasets continue to grow. More challenging to store and process entire datasets locally.

Traditional approach

- Replicating entire datasets increases availability and helps distribute computation.
- We schedule the computation close to the data.
Motivation and Background

Limitations of the traditional approach

- Insufficient number of replicas may delay task execution.
- The computation might require two datasets stored at different locations.
- Fixing these problems requires making new replicas.
- Copying large replicas over a network may take hours or days.
Limitations of the traditional approach

- If data are stored at a single site, we are limited by local storage and local computational resources.

- Even if storage is plentiful at other sites, we can not use it unless we break the dataset into smaller pieces.

- We might need only a portion of the entire dataset.

- Accessing the required subset could be cheaper than moving the dataset or even the computation.
Motivation and Background

Possible solutions to these limitations:

- Create partial replicas — copies of portions of the original dataset.
- Provide fast subset access to spatial datasets.
- Allow computation to be scheduled on machines not local to data.
- Both data and computation are distributed in the system.
Motivation and Background

A spatial dataset associates data values with locations in an n-dimensional domain.

A **storage model** is especially important for spatial data.

- How to map n-D data to 1D disk files?

- We can make storage models for different types of spatial data.

  - e.g. n-D Arrays, unstructured points, space-filling curves
Motivation and Background

Partial Spatial Replicas

represent spatial subsets of the larger data volume.

associated with metadata such as subset bounds (MBR), physical address, logical file name (LFN), storage organization, etc.

MBR = \{X_{min}, Y_{min}, X_{max}, Y_{max}\}, for a 2D example.

Many replicas may cover a same region.

Figure 1, Partial Spatial Replicas
Related Work

Distributed Spatial Computation and Visualization Environment (DISCoVer)

- **Granite:**
  - Efficient access subset of spatial dataset locally or remotely using UDT protocol [Gu and Grossman]
  - Provides a model of how data is stored on disk.

- **Magnolia:** integrates Granite with existing Grid software, and provides spatial replica location service (SRLS) and replica selection.
Related Work

Replica Selection of complete replicas

- Rahman’s two selection techniques in grid: k-Nearest Neighbor based approach and neural network approach.
- Vazhkudai et al. presents a high-level replica selection service, using replica location and user preferences and ClassAds mechanism.
- Li’s PU-DG Optibox package to facilitate parallel download from a ranked list of grid nodes.
- Zhao presents GRESS is based on Open Grid Service Architecture.
- Gfarm grid file system supports replica selection for complete replicas.
Replica Selection of partial replicas

Chang et al. present a fragmented replica selection algorithm, by assuming each block has a same downloading time.

Narayanan et al. describe GridDB-Lite and a runtime framework to address the partial spatial replica replication problem.

Weng et al. present a partial replica selection algorithm for serving range queries on multidimensional datasets. The complexity is $O(n^2)$, where $n$ is the number of chunks intersected with a range query.
Related Work

Distinctions

- Partial Spatial Replicas have arbitrary shape and size.
- Not limited to uniform chunks.
- We do not assume all replicas have the same network cost.
- Suitable for grid and cloud computing.
- Our work takes into account the filesystem caching and prefetching.
- Our selection algorithm has lower complexity.
Outline

- Motivation and Background
- Related Work
- Models
- Partial Spatial Replica Selection
- Experiments
- Conclusion
First, we have to identify a set of partial replicas that intersect with a spatial query.

For a given subregion, many replicas could answer the query.

Second, we have to choose a subset of these partial replicas so that the performance is maximized.
Models

The Replica Selection Problem for Spatial Datasets
Models

To estimate the cost of retrieving the subset of replicas and choose a combination to read requires several models.

- Storage Model
- Access Pattern
- Device Model
- Modeling Replica Transfer Time
The Rod Storage Model is used to store n-D files in linear order on disk in figure 2.

“Rod” or “Space Rod” refers to a sequence of data elements that are contiguous both on disk and in the index space.

“Query rod” is the portion that intersects with a specific subset query within a space rod.
The Rod Storage Model is used to store n-D files in linear order on disk in figure 2.

“Rod” or “Space Rod” refers to a sequence of data elements that are contiguous both on disk and in the index space.

“Query rod” is the portion that intersects with a specific subset query within a space rod.
A **storage ordering** denotes the order in which data is stored on disk.

The **rod axis** and the **slice axis** denote the direction of the rods and slices.

Different storage orderings may result in different access costs, given the same query and data space.
Rod Storage Model

Storage ordering
\{0,1,2\}

Storage ordering
\{1,2,0\}

Storage ordering
\{2,0,1\}
Rod Storage Model

Storage ordering 
\{0,1,2\}  

Storage ordering 
\{1,2,0\}  

Storage ordering 
\{2,0,1\}  

0 1 2 0 1 2 0 1 2
Rod Storage Model

Storage ordering
{0,1,2}

Storage ordering
{1,2,0}

Storage ordering
{2,0,1}
Access Patterns represents the series of read transactions made to the 1-D file underlying the spatial dataset.

Generally denoted as $A = \{(s_0, l_0), (s_1, l_1), ..., (s_n, l_n)\}$, where $s$ indicates the stride and $l$ is the length of a read transaction.

But with rod storage model, subset query access patterns can be represented using $s_{slice}$, $s_{rod}$, $l_{qrod}$, $l_{slice}$ and $n_{slice}$. 
Models — Access Pattern

(a) A 3D dataset with three slices and Storage ordering \( \{0,1,2\} \). Assume the blue region is the 3D subset query, may or may not penetrate through all slices.

(b) One of slices in dataset and the 2D intersection with the subset query.

(c) 1D file storage on hard disk. The red arrow is the file pointer. Assume each cell takes one unit of length, the space rod size \( s_{rod} = 8 \) units. The slice size \( s_{slice} \) is 64 units. The read length covered by range query within one space rod is \( l_{qrod} = 2 \) units. Number of query rods on each slice \( l_{slice} = 4 \) and number of slices covered by the query \( n_{slice} \) is 3.
Models — Access Pattern

(a) A 3D dataset with three slices and Storage ordering \{0,1,2\}. Assume the blue region is the 3D subset query, may or may not penetrate through all slices.

(b) One of slices in dataset and the 2D intersection with the subset query.

(c) 1D file storage on hard disk. The red arrow is the file pointer. Assume each cell takes one unit of length, the space rod size \(s_{rod} = 8\) units. The slice size \(s_{slice} = 64\) units. The read length covered by range query within one space rod is \(l_{qrod} = 2\) units. Number of query rods on each slice \(l_{slice} = 4\) and number of slices covered by the query \(n_{slice} = 3\).
(a) A 3D dataset with three slices and Storage ordering \{0,1,2\}. Assume the blue region is the 3D subset query, may or may not penetrate through all slices.

(b) One of slices in dataset and the 2D intersection with the subset query.

(c) 1D file storage on hard disk. The red arrow is the file pointer. Assume each cell takes one unit of length, the space rod size \(s_{rod} = 8\) units. The slice size \(s_{slice}\) is 64 units. The read length covered by range query within one space rod is \(l_{qrod} = 2\) units. Number of query rods on each slice \(l_{slice} = 4\) and number of slices covered by the query \(n_{slice}\) is 3.
(a) A 3D dataset with three slices and Storage ordering \{0,1,2\}. Assume the blue region is the 3D subset query, may or may not penetrate through all slices.

(b) One of slices in dataset and the 2D intersection with the subset query.

(c) 1D file storage on hard disk. The red arrow is the file pointer. Assume each cell takes one unit of length, the space rod size $s_{rod} = 8$ units. The slice size $s_{slice}$ is 64 units. The read length covered by range query within one space rod is $l_{qrod} = 2$ units. Number of query rods on each slice $l_{slice} = 4$ and number of slices covered by the query $n_{slice}$ is 3.
(a) A 3D dataset with three slices and Storage ordering \{0,1,2\}. Assume the blue region is the 3D subset query, may or may not penetrate through all slices.

(b) One of slices in dataset and the 2D intersection with the subset query.

(c) 1D file storage on hard disk. The red arrow is the file pointer. Assume each cell takes one unit of length, the space rod size \( s_{rod} = 8 \) units. The slice size \( s_{slice} \) is 64 units. The read length covered by range query within one space rod is \( l_{qrod} = 2 \) units. Number of query rods on each slice \( l_{slice} = 4 \) and number of slices covered by the query \( n_{slice} \) is 3.
(a) A 3D dataset with three slices and Storage ordering \{0, 1, 2\}. Assume the blue region is the 3D subset query, may or may not penetrate through all slices.

(b) One of slices in dataset and the 2D intersection with the subset query.

(c) 1D file storage on hard disk. The red arrow is the file pointer. Assume each cell takes one unit of length, the space rod size $s_{rod} = 8$ units. The slice size $s_{slice}$ is 64 units. The read length covered by range query within one space rod is $l_{qrod} = 2$ units. Number of query rods on each slice $l_{slice} = 4$ and number of slices covered by the query $n_{slice}$ is 3.
(a) A 3D dataset with three slices and Storage ordering \{0,1,2\}. Assume the blue region is the 3D subset query, may or may not penetrate through all slices.

(b) One of slices in dataset and the 2D intersection with the subset query.

(c) 1D file storage on hard disk. The red arrow is the file pointer. Assume each cell takes one unit of length, the space rod size \(s_{\text{rod}} = 8 \) units. The slice size \(s_{\text{slice}} = 64 \) units. The read length covered by range query within one space rod is \(l_{\text{qrod}} = 2 \) units. Number of query rods on each slice \(l_{\text{slice}} = 4 \) and number of slices covered by the query \(n_{\text{slice}} = 3 \).
(a) A 3D dataset with three slices and Storage ordering \{0,1,2\}. Assume the blue region is the 3D subset query, may or may not penetrate through all slices.

(b) One of slices in dataset and the 2D intersection with the subset query.

(c) 1D file storage on hard disk. The red arrow is the file pointer. Assume each cell takes one unit of length, the space rod size \(s_{rod} = 8\) units. The slice size \(s_{slice}\) is 64 units. The read length covered by range query within one space rod is \(l_{qrod} = 2\) units. Number of query rods on each slice \(l_{slice} = 4\) and number of slices covered by the query \(n_{slice}\) is 3.
Models — Access Pattern

(a) A 3D dataset with three slices and Storage ordering \{0,1,2\}. Assume the blue region is the 3D subset query, may or may not penetrate through all slices.

(b) One of slices in dataset and the 2D intersection with the subset query.

(c) 1D file storage on hard disk. The red arrow is the file pointer. Assume each cell takes one unit of length, the space rod size \( s_{rod} \) =8 units. The slice size \( s_{slice} \) is 64 units. The read length covered by range query within one space rod is \( l_{qrod} \) = 2 units. Number of query rods on each slice \( l_{slice} \) = 4 and number of slices covered by the query \( n_{slice} \) is 3.
Models — Access Pattern

(a) A 3D dataset with three slices and Storage ordering \{0,1,2\}. Assume the blue region is the 3D subset query, may or may not penetrate through all slices.

(b) One of slices in dataset and the 2D intersection with the subset query.

(c) 1D file storage on hard disk. The red arrow is the file pointer. Assume each cell takes one unit of length, the space rod size \( s_{rod} = 8 \) units. The slice size \( s_{slice} \) is 64 units. The read length covered by range query within one space rod is \( l_{qrod} = 2 \) units. Number of query rods on each slice \( l_{slice} = 4 \) and number of slices covered by the query \( n_{slice} \) is 3.
Models — Access Pattern

(a) A 3D dataset with three slices and Storage ordering \( \{0,1,2\} \). Assume the blue region is the 3D subset query, may or may not penetrate through all slices.

(b) One of slices in dataset and the 2D intersection with the subset query.

(c) 1D file storage on hard disk. The red arrow is the file pointer. Assume each cell takes one unit of length, the space rod size \( s_{rod} = 8 \) units. The slice size \( s_{slice} \) is 64 units. The read length covered by range query within one space rod is \( l_{qrod} = 2 \) units. Number of query rods on each slice \( l_{slice} = 4 \) and number of slices covered by the query \( n_{slice} \) is 3.
(a) A 3D dataset with three slices and Storage ordering {0,1,2}. Assume the blue region is the 3D subset query, may or may not penetrate through all slices.

(b) One of slices in dataset and the 2D intersection with the subset query.

(c) 1D file storage on hard disk. The red arrow is the file pointer. Assume each cell takes one unit of length, the space rod size $s_{rod} = 8$ units. The slice size $s_{slice}$ is 64 units. The read length covered by range query within one space rod is $l_{qrod} = 2$ units. Number of query rods on each slice $l_{slice} = 4$ and number of slices covered by the query $n_{slice}$ is 3.
A storage model indicates the number of device transactions a subblock query will require (access pattern), while a device model should indicate the cost of each of those transactions.

The device model for replica selection only needs to be accurate enough to let us choose replicas well.
We don’t keep track of cache contents.

We proceed in the forward direction when reading, so we never revisit pages.

We do keep track of whether the next page to be accessed has been prefetched.

Filesystem prefetching and caching boosts performance by 2 or 3 times.
Models — Device Model

The device model consists of two components: raw device model and cache model.

- cache model takes an access pattern as input and returns an estimate of the time required to perform I/O.

- cache model uses the raw device model to estimate access time to the disk device.

- raw device model implements two queries: DateTime(readlength) and Latency(stride).
Models — Device Model

Raw Device Model

- **DateTime**(readlength) returns the time cost of retrieving that number of bytes via available device bandwidth, given the length of a read trans.

- **Latency**(stride) returns a time cost associated with making the device move that distance before reading the next piece of data.

- We experimentally determined the Latency function.
We experimentally step through a big file with strides ranging from 1 byte to over 5MB, when filesystem cache was disabled.

We used piecewise linear regression to approximate the stride—latency function in figure below.

![Figure, Relation between stride size and disk latency.](image)
Cache Model

- Takes an access pattern as input and returns an execution time.
- Considering a single replica here, to answer a subset query, data server has to traverse through all slices covered by the query.
- The cache model has to model filesystem prefetching behavior.
Models — Device Model

Filesystem Prefetching Behavior

- determined by system settings and whether an application is accessing the file **sequentially**.

- If program currently reads block x and the last read was from either x or x - 1, filesystem treat the current access as sequential.

- If current access is sequential, filesystem will double the amount of data prefetched up to a maximum number of sectors (called prefetch size). If not, the prefetch size will cut in half down to a minimum prefetch size.
Figure, 1-D file storage on hard disk and filesystem prefetching behavior. We assume that each system block consists of 5 units of length. From 0 through offset 63, we have 4 file reads. But the last read at $B_9$ is Non-Sequential, while two reads in the middle are considered sequential. We have to model how many space rods the filesystem prefetched into buffer for each disk read.
Figure, 1-D file storage on hard disk and filesystem prefetching behavior. We assume that each system block consists of 5 units of length. From 0 through offset 63, we have 4 file reads. But the last read at B₉ is Non-Sequential, while two reads in the middle are considered sequential. We have to model how many space rods the filesystem prefetched into buffer for each disk read.
Figure, 1-D file storage on hard disk and filesystem prefetching behavior. We assume that each system block consists of 5 units of length. From 0 through offset 63, we have 4 file reads. But the last read at $B_9$ is Non-Sequential, while two reads in the middle are considered sequential. We have to model how many space rods the filesystem prefetched into buffer for each disk read.
Figure, 1-D file storage on hard disk and filesystem prefetching behavior. We assume that each system block consists of 5 units of length. From 0 through offset 63, we have 4 file reads. But the last read at B₉ is Non-Sequential, while two reads in the middle are considered sequential. We have to model how many space rods the filesystem prefetched into buffer for each disk read.
Figure, 1-D file storage on hard disk and filesystem prefetching behavior. We assume that each system block consists of 5 units of length. From 0 through offset 63, we have 4 file reads. But the last read at B₉ is Non-Sequential, while two reads in the middle are considered sequential. We have to model how many space rods the filesystem prefetched into buffer for each disk read.
Models — Device Model

Cache Model

To model prefetching behavior, we compute an Average Prefetch Count (APC) in equation below, corresponding to the average number of space rods that are prefetched by the filesystem within one slice.

\[
APC = \begin{cases} 
1 & \text{if } (s_{rod} - l_{qrod}) \geq 2b, \\
\frac{P_{max} d}{s_{rod}} & \text{if } (s_{rod} - l_{qrod}) \leq b \\
\text{simulate()} & \text{if } b < (s_{rod} - l_{qrod}) < 2b
\end{cases}
\]

\(S_{rod} - l_{qrod}\) is the gap between the end of one read and the beginning of the next one. \(d\) is size of disk sector, \(b\) is the filesystem block size. \(P_{max}\) is the maximum number of sectors prefetched by system.
Models — Device Model

Cache Model

To compute an estimate of the total disk costs \( T_d \) associated with reading the subset volume \( s \), we use equation 1, where \( C_0 \) is the cache overhead and computed using equation 3.

\[
T_d(s) = n_{slice} \times \left[ DiskLatency(s_{slice}) + \frac{l_{slice} - 1}{APC} \times (DiskLatency( APC \times s_{rod} ) + C_o ) \right] + DataTime( size(s) ) \tag{1}
\]

\( k \) in equation 3 equals to 1.6 in our experiments.

\[
C_o = \begin{cases} 
0.1 & \text{if } APC < 2, \\
\frac{1}{k} & \text{otherwise}
\end{cases} \tag{3}
\]
To model the network cost and the behavior of servers and clients in a distributed system, we make several assumptions.

- Each server stores all data files on a single storage device,
- Both network bandwidth and latency are constant, and are recently measured.
- All servers transferring data to a single client have an equal share of client bandwidth. If we add up those shares, the sum will not be greater than the bandwidth that would be consumed by a single server.
- We use the UDT protocol that shows a similar property as with this assumption.
The modeled network cost \( T_n(s \subseteq r) \) to transfer the subset \( s \) or a replica \( r \) residing on \textit{Server} to a \textit{Client} is shown in equation 4.

\[
T_n(s) = \frac{\text{size}(s)}{\text{NetworkBandwidth}(	ext{Client}, \text{Server}) + \text{NetworkLatency}(	ext{Client}, \text{Server})}
\]

Total cost \( T(s) \) consists of the disk access cost and network cost in equation 5.

\[
T(s) = T_d(s) + T_n(s);
\]  

\( T(\text{host}) \) is the transfer time for a collection of subsets \( H_{\text{host}} = \{s_0 \subseteq r_0, s_1 \subseteq r_1, ..., s_n \subseteq r_n \} \) on a host, shown in equation 6.

\[
T(H_{\text{host}}) = \sum_{i=0}^{n} T(s_i);
\]  

The model assumes no overlap transfer costs of various subset on a single host.
To compute the overall transfer time $T(Q)$ of a spatial query $Q$, we must first identify the set of hosts $H = \{H_0, H_1, ..., H_m\}$ containing the replicas that will be used to satisfy the query. $T(Q)$ is written in equation below.

$$T(Q) = \max(T(H_0), T(H_1), ...T(H_m))$$

We assume that the various hosts involved in a query can work in parallel and independently.

The host that takes the most time to complete the query is called the bottleneck. $T(Q) = T(\text{bottleneck})$.

The replica selection algorithm requires minimizing $T(\text{bottleneck})$ by distributing loads.
Partial Spatial Replica Selection

Replica Selection Algorithm chooses a subset of replicas among all intersected replicas, so that the total transfer time for that subset is minimized.

We made simplifying assumption that if two replicas overlap, they have the same MBR. But such replicas might be represented using different storage orderings, and be on different grid nodes.
Partial Spatial Replica Selection

Given a subset $s_i \subseteq R_i$ of replica $R_i$ stored on Host$_i$, the goodness value of $s_i$ can be calculated using equation 8.

$$goodness(s_i \subseteq R_i) = \frac{Size(s_i) \times Network\text{Bandwidth}(Client_i, Host_i) \times Disk\text{Bandwidth}(Host_i)}{Num\text{Reads}(R_i) \times Network\text{Latency}(Client, Host_i) \times Disk\text{AverageSeekTime}(Host_i)}$$ (8)
INPUT: Query Q; a set of partial replicas S that intersected with Q.
OUTPUT: S', a subset of S, so that S' can answer the query Q and the
transfer time of S' is minimized.

1. S' = {} //empty set
2. St = Sort(S) //sort S according to goodness of replica in descending
order
3. initialize(C) //use C to store a collection of selected replicas
4. while St not empty
5.   Replica r = getFirstElement(St)
6.   St = St - r
//Let u be the replica in C that matches r.
//Note that C contains no more than one match for r.
7.   u = C.match(r)
8.   if u does not exist
9.     C.insert(r)
10.    S' = S' + r
11.   else
12.     S* = S' - u // temporary set S*
13.    S* = S* + r
14.    if ( T(S') > T(S*) ) or ( T(S') == T(S*) and T(r) < T(u) )
15.       C.delete(u)
16.       C.insert(r)
17.       S' = S*
18.     end if
19.   end else
20. end while

Step 14 ensures the bottleneck host changes as the algorithm progresses
and the load is spread among many nodes.
For the collection C in step 3, we used an existing R*-tree implementation. We call it the R-tree based replica selection implementation.

Currently, the R-tree is not strictly necessary. We could use a hash table for C which reduces the complexity from $O(n \log n)$ to $O(n)$.

But in our future work, we will handle the more general scenario where replicas could be arbitrarily overlapped, which requires a spatial index method.
Experiments

We test in the Distributed Research Testbed (DiRT), a multi-site instrument spread across five US universities.

The characteristics of each grid node in DiRT are described in table 1.

We use a client machine at the University of Mississippi, the UDP bandwidth between the client and various servers are shown in table 2.
Experiments

We used Granite Scientific Database as our data server, which in turn uses the UDT protocol for data transfer, developed by Gu and Grossman.

Table 1 Grid Node Characteristics

<table>
<thead>
<tr>
<th>OS</th>
<th>Processor</th>
<th>Cores</th>
<th>Memory</th>
<th>Hard Disk</th>
<th>File System</th>
<th>UDT</th>
<th>Java version</th>
<th>Globus Toolkit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux 2.6.18</td>
<td>Intel Xeon 2.40GHz</td>
<td>16</td>
<td>24G</td>
<td>1TB</td>
<td>ext3</td>
<td>v4.10</td>
<td>SE 1.6.0_23</td>
<td>v5.0.3</td>
</tr>
</tbody>
</table>

Table 2 Network Parameters Between A Client At Mississippi and Various Data Servers

<table>
<thead>
<tr>
<th></th>
<th>Univ. Mississippi</th>
<th>Univ. Florida</th>
<th>Univ. Chicago</th>
<th>Univ. Notre Dame</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>746Mbit/s</td>
<td>735Mbits/s</td>
<td>724Mbits/s</td>
<td>718Mbits/s</td>
</tr>
<tr>
<td>Latency</td>
<td>0.28ms</td>
<td>52ms</td>
<td>41ms</td>
<td>95ms</td>
</tr>
</tbody>
</table>
Experiments

Experiments are designed with two goals.

- Verify that the device model and storage model can estimate the disk access cost accurately.
- Verify that our selection algorithm can choose the right set of replicas with the help of the proposed device model and storage model.

- We compared with selection solutions found by exhaustive search.
- Exhaustive search can find the real optimal and worse selection solution.
Experiments

Verifying Device Model and Storage Model in First Scenario

Dataset with space rod size that is a multiple of filesystem block size. MBR={0,0,0,2047, 2047,2047}. Each point in the data domain has a double float type, which takes 4 bytes.

Verifying Device Model and Storage Model in Second Scenario

Dataset with space rod size that is NOT a multiple of filesystem block size. MBR={900,750,900,1500, 2750,3399}. 
Storage and Device Model Verification under first Scenario. Each query increments by 50 the largest coordinate of the previous query MBR on axis 2, to change the query access rod size. The space rod size is a multiple of page size of 4KB.
storage and device model verification under second scenario. each query increments by 50 the largest coordinate of the previous query mbr on axis 2, in order to change the query access rod size. the space rod size is not a multiple of page size of 4kb.
All results show a close match except for a few errors under the second scenario, where space rod size is NOT a multiple of page size.
Experiments

The error is due to the fact that we only have two values for the cache overhead $C_0$ in equation 3, using a simple step function to relate $C_0$ to APC.

- $C_0 = 0.1$ when $APC < 2$ corresponds to when prefetching is not active.
- $C_0 = 1.6$ when $APC >= 2$ for an active prefetching.

The real disk behavior shows more gradual behavior in which prefetching becomes increasingly active over a range of APC values.

The big error in figure 4(b) for that query has a APC=2.03, but the real filesystem prefetching has not activated yet. The model in turn used a wrong $C_0$.

$$C_o = \begin{cases} 0.1 & \text{if } APC < 2, \\ k & \text{otherwise} \end{cases} \quad (3)$$
Experiments

Verifying Replica Selection Algorithm

Two purposes when verifying our algorithm

First, we have to verify the selection solution found by our algorithm is good compared with optimal solution.

Second, we have to verify the estimated transfer time computed by our models should match the real transfer time in a distributed environment.
Experiments

Verifying Replica Selection Algorithm

- The MBRs of all partial replicas are randomly generated by a recursive random bisection algorithm.

- For each MBR, we create three different replicas, each with a different storage ordering: \{0, 2, 1\}, \{1, 0, 2\} or \{2, 1, 0\}. 
Experiments

Verifying Replica Selection Algorithm

Dataset MBR in 4-node tests is \(\{0, 0, 0, 8192, 8192, 4096\}\). Number of replicas in grid is 3072, total size is 3.07TB.

Dataset MBR in 8-node tests is \(\{0, 0, 0, 8192, 8192, 8192\}\). Number of replicas in grid is 3072, total size is 6.14TB.

The subset queries are generated randomly. We randomly generate their location in the domain and their dimensions along each axis between 10 to 700.
Experiments

Real time cost of solution using R-tree based replica selection

Real time cost of R-tree based, optimal and worst solution using 4 nodes
Experiments

Real time cost of solution using R-tree based replica selection

Real time cost of R-tree based, optimal and worst solution using 4 nodes

Real time cost of solutions found by our algorithm are close to the optimal solution.
Experiments show that our selection algorithm can find a solution that has excellent performance compared with optimal solution.
Experiments

Real time cost of solution using R-tree based replica selection

Solution found by our algorithm is as good or better than 98% of solutions found by exhaustive search.
Performance is on average always at least 91% and 93.4% of the real time cost of the optimal solution in 4 and 8 nodes tests.
Experiments

Beside the real world tests with 4 and 8 grid nodes, we also ran a 64-node simulation with 546000 replicas.

- We observed a similar performance as in 8-node tests subset query.
- Most of these replicas are eliminated by the intersection test. [Tian11]

Execution of the R-tree based replica selection implementation takes around 3 seconds for 6000 replicas.
Experiments

- Estimated time cost for optimal solution (bar lower end) and worst solution (bar upper end)
- Real time cost for optimal solution (bar lower end) and worst solution (bar upper end)

Real time cost compared with estimated time cost using 4 nodes
Experiments show that estimated time cost matches well with the real time cost for optimal solution.
Experiments

Experiments show in most cases the estimated time cost matches well with real time cost for optimal solutions (lower bar end in figure). For worst solutions, they are less exactly matched, but their fluctuation trends are similar. Matching the worst solution is not a goal of replica selection.
We observed an average 3% of model failure during tests, Q5 and Q21 shown in figure.

The failure is not because of the increased number of grid nodes, but caused by our device model error.
Conclusion and Future Work

We describe a fast replica selection algorithm that provides load balancing and take file storage organization, filesystem prefetching and hardware performance into account using separate models.

Performance of the solution found by our algorithm is on average always at least 91% and 93.4% of the real time cost of the optimal solution in 4 and 8 nodes tests.

The complexity of our algorithm is lower.
Conclusion and Future Work

In future research:

- We will add more storage models and multiple storage devices which allows to accommodate a wider variety of databases.
- We may replace our static network model with a grid Monitoring and Discovery System (MDS).
- We will address the error in our device model.
- We will address the more general replica selection scenario, in which replicas overlap in an arbitrary way.
Acknowledgment

This work was supported by the National Science Foundation under grants CCF-0541239 and CRI-0855136.

Special thanks to the anonymous reviewers for their valuable comments.