A Configurable Application View Storage System

Sibel Adali
Rensselaer Polytechnic Institute
Overview of talk

• Overview
• Related work
• Motivation - information integration
• Storage management
  - View re-use
  - View eviction
• Experimental results, preliminary findings
• An example view management problem
• Research problems
CAVES

- General purpose storage management system for networked applications which includes a “storage shell” that incorporates
  - Programmable storage management protocols
  - Complex view re-use options
  - Dynamic tuning of protocols to actual system loads
  - Verification modules for the effectiveness of the overall system
CAVES - challenges

- Unit of storage is not uniform → views of varying sizes
- Storage system may need to serve multiple types of queries with different characteristics → shared storage pool
- Characteristics of actual requests are unpredictable → unknown storage policies
- The cost parameters may change from application to application → different optimization problems
Storage Management

- Client-side caching of views – instead of tuple based caching [Franklin et. al.]
- Semantic caching [Franklin, Chu, Godfrey, ...]
- OLAP/ Data Cube applications
  - WATCHMAN [Scheuermann]
  - DYNA-MAT [Kotidis]
  - Proxy-server architectures for OLAP [Kalnis, Papadias]
CAVES - challenges

1. Are any common tasks a storage management system has to perform?
2. How much does an application programmer have to know to develop an effective storage management system?
Networked applications

• Require data from distributed sources
  - Sources are distributed without the application in mind

• Sources are possibly autonomous, heterogeneous
  - No control over their internal data model
  - No agreements to notify the users of updates

• Sources are possibly integrated using views: local point of view of the data
Networked Applications

User request

Determine which sources to query

Translate queries to source syntax and send requests

Application Server

Wrapper/Translator

Wrapper/Translator

Wrapper/Translator

Data Server 1

Data Server 2

Data Server k

Send queries to sources

Configurable Application View
Storage System
Networked Applications

- Wrapper/Translator
- Data Server 1
- Data Server 2
- Data Server k

Query result = view
Combine the results and respond to the user
Receive response from sources

Configurable Application View
Storage System
Information integration

- Municipal network connecting multiple agencies to the office of the mayor
  - Population in different regions [CENSUS BUREAU]
  - Property boundaries [STATE TAX OFFICE]
  - Locations of traffic patterns, congestions [CITY]
  - Locations of major incidents [POLICE]
  - Dates/times of major events [LOCAL ORGANIZATIONS]
Information Integration

- Map the information of interest from each office to a “local schema”

Population(Neighborhood, Popl, F%, M%, PC%, C%, H%, Ages0-20, Ages20-40, Ages40-60, Ages60-)

Property(TaxID#, Neighborhood, Area, Boundary:Polygon)

CaseReport(Day, Type, Neighborhood, TotalArrests, TotalCases)
Information Integration

• What each attribute means is defined by the “local schema”
• How to extract this information is defined by the “integration scheme”
• Queries are posed to the application server that operates on the local schema
  - Which neighborhoods have the highest population of ages 20 and below per km²?
  - Which neighborhoods have been showing increasing cases of pothole related murders?
  - What is the distribution of education across neighborhoods? [RESULT: map]
CREATE VIEW V1(A,B)
AS SELECT PT.neighborhood,
    sum(area(intersect(PT.boundary, MP.boundary)))
FROM property PT, maps MP
WHERE MP.neighborhood=PT.neighborhood
GROUP BY PT.neighborhood

CREATE VIEW V2(A,B,C,D)
AS SELECT C.day, C.neighborhood,
    avg(C.totalarrests), stddev(C.totalarrests)
FROM casereports C
GROUP BY cube(C.day, C.neighborhood)
Challenges

- Most data manipulation operations are very costly at the server
- Views are probably costly to transmit, too large to replicate
- New economies:
  - Unpredictable response time
  - Cost of accessing services /not cost of maintaining servers (pay-per-view)
  - Consistency is not a major concern → archival data
  - Exact values are not always necessary → approximate answers
Networked Applications

Can we reuse any of the existing views?
**View reuse**

- An existing view, fragment can be used to answer a new query:
  - Existing view: arrest data for the last six weeks
  - New query: arrest data for the last two weeks for ‘South Troy’
  - Reuse: **select** on the existing view

- Existing view: traffic congestion data for the last ten days for ‘Niskayuna’
- New query: traffic congestion summary for the last week for ‘Niskayuna’
- Reuse: **aggregate** on the existing view
View reuse

- Existing: an image representing a map
- New: a smaller area completely contained within the existing map
- Reuse: crop the existing map/image
View reuse

- View reuse methods are being developed for all applications
  - Answering queries using views
  - Spatial indexing mechanisms
  - Each method is application specific, implemented by an AP and linked to CAVES
- Integrate view reuse methods with storage server
  - Instead of naïve look-up, invoke view reuse methods
  - Costly look-up methods may not always pay off → cost based
To reuse or not to reuse

Request Q

Answer to Q

CAVES

View reuse methods

Retrieve Q from the server

Find all appropriate V
Compare costs
Pick the lowest cost V’

Answer to Q using V’

Keep statistics about the use of V’

Local storage

How to compare the cost of reuse to cost of retrieving from the server?
- Lower bound based on the expected response time and estimated size
- Always connect the server as a back-up plan
To reuse but...

- View consistency is a major concern for some applications
  - Keep a deadline for each “view type”, and remove it after the deadline [our current approach]
  - Refresh views periodically by polling the servers [autonomous data servers]
    - Predict the rate of change based on observations
    - Invoke incremental view maintenance methods if they exist
  - Obtain leases on objects from servers, servers notify updates [dedicated data servers]
Statistics to keep

- For each view that is being stored, keep a number of statistics
  - Id, location on disk
  - Description → used to determine possible reuse
  - Performance statistics

- Size
- ComputationCost
- ServerTransferRate
- Age

LastRef
Hits
FrequencyOfUse
PercentageOfUse

Do not change by reuse
Change by reuse
Statistics

• Each statistics has a
  - Name
  - Method for initialization
  - <optional> Method for update

• Each application decides which statistics to keep
  - A number of statistics are provided by the system
  - They can be added by an AP

Incremented hits by 1
Poll servers
Storage Management

- Storage management is an optimization problem
  - A limited amount of storage space
  - Keep in storage items of highest value
- How to define the “value” of an item?
  - Use statistics to write functions that mimic the value objective → “order formula”
  - Use weights to determine the “importance” of an order formula in determining the value → “weight vector”
  - Weights represent a tunable parameter
Order formulae

Higher value represent higher priority → more desirable

\[ F_1 = \frac{\text{computationCost}}{(\text{size} \times \text{age})} \] \text{[OLDESTFIRST]}
\[ F_2 = \text{hits} \] \text{[LU]}
\[ F_3 = \frac{1}{\text{serverTransferRate}} \] \text{[FNSF]}
\[ F_4 = \text{lastRef} \] \text{[LRU]}
\[ F_4 = \frac{\max(k, \text{hits})}{(\text{lastRef} - \text{lastToKRef})} \] \text{[LFU]}
\[ F_5 = F_1 \times F_4 \] \text{[LFUwSIZE]}
Priority

- No order formula alone leads to better performance for all applications
  - Hits is not a good measure for items requested periodically
  - Computation cost and size alone are not good measures if they do not vary greatly among views
  - Computation cost may vary depending on the network and server load
Priority

- The system is usually biased against a new view with no hits
  - Use computation-cost/size for these views and then switch to the regular priority function [WATCHMAN]
  - Our solution:
    \[ F_1 = \frac{\text{computationCost}}{\text{size} \times \text{age}} \]
    \[ F_2 = \text{hits} \]
    
    Use: \[ w_1 \times F_1 + w_2 \times F_2 \]
Storage management with priority

• Given a set of order formula $F_1, \ldots, F_k$, a weight vector $W = <w_1, \ldots, w_k>$, the priority of each view $V$ is given by
  \[ P(V) = w_1 \times F_1(V) + \ldots + w_k \times F_k(V) \]
• If we need to remove a view to admit a new one, then we need to pick the one with the current lowest priority
• How can we do this quickly?
  - Keep all views in sorted order of their current priority using a priority queue
Storage management with priority

\[ \text{AvailableSpace} = \text{SS} - \sum\{ \text{size}(V') \mid V' \text{ in storage} \} \]

\( V \): new view just admitted to the cache

Pick views \( V_1, \ldots, V_k \) such that

\[ P(V_1) + \ldots + P(V_k) \] is minimal over all views that free up \( \text{size}(V) - \text{AvailableSpace} \)

A version of the knapsack problem (NP-complete)
Storage management with priority

- Greedy heuristic:
  - Pick the items in the order of their priority until sufficient space is freed
  - Would like to choose bigger views first → smaller the number of views implies smaller total priority
  - Use unit priority instead of actual priority
    \[
    \text{unitPriority} = \frac{\text{priority}}{\text{sizeFactor}}
    \]
  - Small size factor → view size has no effect
  - Large size factor → priorities lose importance
Successful look-up

- **CAVES**
  - Request Q
  - Respond

- **View reuse methods**
  - Look up view descriptions
  - Update view statistics for V

- **Priority Queue**
  - Delete V, re-insert at its correct priority

- **Local storage**
  - Read V

Configurable Application View Storage System
Unsuccessful look-up

- **CAVES**
  - Request Q
  - Respond
- **View reuse methods**
  - Retrieve V from server
- **Priority Queue**
  - Compute statistics for V
  - Eligible to enter?
- **View management**
  - Remove lowest priority views
  - Store V
- **Local storage**
Priority queue

- In this model, we update the priority queue only when a view is updated
- How about
  \[ F_1 = \frac{\text{computationCost}}{\text{size} \times \text{age}} \]
- Too costly to resort the queue regularly
- Keep two queues
  - Priority queue (PQ)
  - Timed output queue (TOQ) → schedule when the priority of certain views should be recalculated
  - For each view, create \( k \) ghosts in TOQ
Weight change rules

- It is unlikely that the same weights will maximize performance over all workloads.
- Observe statistics over all order formulas in the system:
  - Max, min, avg of values these functions take
  - Standard deviation of values in the PQ
- Use rules to define when the weights should change:
  \[ \text{Stddev}(F1)/(\text{Max}(F1)-\text{Min}(F1)) < .2 \rightarrow [.2, F1] \]
Testing and Results

- The CAVES system is implemented in Java
  - Allows the specification of order formula and weight change rules from a system specification (XML document)
- A simulation model is implemented in C
- The simulation model and the actual system has been aligned for a number of test cases (30 min runs) on an Ultra10
Simulations

- Mixed workloads over five different query types
  - A type is defined by “size” of the resulting view, and the overall search time
  - An indexed selection is smaller in search time, varies in size [equality – range search], types 0, 1, 2
  - An aggregation query is very large in search time, typically small in size, types 3, 4
  - Each type has a dedicated server, the network rate varies greatly for types 1 and 3.
## Simulations

<table>
<thead>
<tr>
<th>Qtype</th>
<th>#Desc</th>
<th>Size(K)</th>
<th>FF</th>
<th>CoC</th>
<th>W0</th>
<th>W1</th>
<th>W2</th>
<th>W3</th>
<th>W4</th>
<th>W5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10000</td>
<td>10-500</td>
<td>3-10</td>
<td>1-2</td>
<td>20</td>
<td>6</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5000</td>
<td>175-200</td>
<td>90-100</td>
<td>2-2</td>
<td>20</td>
<td>47</td>
<td>5</td>
<td>40</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5000</td>
<td>175-200</td>
<td>90-100</td>
<td>2-2</td>
<td>20</td>
<td>47</td>
<td>5</td>
<td>10</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2000</td>
<td>10-50</td>
<td>50-100</td>
<td>2-3</td>
<td>20</td>
<td>0</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1000</td>
<td>10-50</td>
<td>100-200</td>
<td>4-5</td>
<td>20</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>90</td>
</tr>
</tbody>
</table>
Simulations

- Tests with different state space of each query type [possible queries of a specific type] and the locality [the frequency of each request].
- The requests are assumed to be normally distributed over each type.

View ids

#requests

High range

Mid range

View ids

Configurable Application View Storage System
Simulations

- Functions $f_1 = \text{computationCost}/(\text{size} \times \text{age})$, $f_2 = \text{hits}$, $f_3 = 1/\text{nr}$
- Measured performance for runs of $10^5$ seconds
- Varied a number of factors
  - Mean arrival time for requests [1 every 1, 5, 10 secs]
  - Mean time, standard deviation for network rate changes
  - Storage space
  - Weight combinations
- Measure Speedup = $\frac{\text{TimeWithoutCaves}}{\text{TimeWithCaves}}$
Results

• The optimal and near optimal weight cases vary from workload to workload, even for the same workload in different locality cases.

• The performance across cache sizes does not vary greatly across different weights cases.

• Average views in cache do not always predict performance.
Results

• Functions:
  - Hit ratio \( (f_2) \) predicts performance, but it does not always lead to the optimal performance.
  - Network rate \( (f_3) \) is usually a factor in all optimal cases.
  - Size and computation cost \( (f_1) \) is not always the best predictor of performance.
    • For workload 1, best performance always sets \( w_1=0 \)
    • For workload 5, best performance sets \( w_1>0 \).
Weights: $w_1=<0,0,1> w_2=<0,1,0> w_3=<1,0,0> w_4=<0,.2,.8> w_5=<0,.4,.6> w_6=<0,.6,.4> w_7=<0,.8,.2> w_8=<.2,0,.8> w_9=<.4,0,.6> w_{10}=<.6,0,.4> w_{11}=<.8,0,.2> w_{12}=<.2,.8,0> w_{13}=<.4,.6,0> w_{14}=<.6,.4,0> w_{15}=<.8,.2,0> w_{16}=<.2,.2,.6> w_{17}=<.2,.4,.4> w_{18}=<.2,.6,.2> w_{19}=<.4,.2,.4> w_{20}=<.4,.4,.2> w_{21}=<.6,.2,.2>
Weights: \( w_1=<0,0,1> \) \( w_2=<0,1,0> \) \( w_3=<1,0,0> \) \( w_4=<0,2,8> \) \( w_5=<0,4,6> \) \( w_6=<0,6,4> \) \( w_7=<0,8,2> \) 
\( w_8=<2,0,8> \) \( w_9=<4,0,6> \) \( w_{10}=<6,0,4> \) \( w_{11}=<8,0,2> \) \( w_{12}=<2,8,0> \) \( w_{13}=<4,6,0> \) \( w_{14}=<6,4,0> \) 
\( w_{15}=<8,2,0> \) \( w_{16}=<2,2,6> \) \( w_{17}=<2,4,4> \) \( w_{18}=<2,6,2> \) \( w_{19}=<4,2,4> \) \( w_{20}=<4,4,2> \) \( w_{21}=<6,2,2> \)
Weights: \( w_1 = (0, 0, 1) \) \( w_2 = (0, 1, 0) \) \( w_3 = (1, 0, 0) \) \( w_4 = (0, 0.2, 0.8) \) \( w_5 = (0, 0.4, 0.6) \) \( w_6 = (0, 0.6, 0.4) \) \( w_7 = (0, 0.8, 0.2) \) \( w_8 = (0.2, 0, 0.8) \) \( w_9 = (0.4, 0, 0.6) \) \( w_{10} = (0.6, 0, 0.4) \) \( w_{11} = (0.8, 0, 0.2) \) \( w_{12} = (0.2, 0.8, 0) \) \( w_{13} = (0.2, 0.4, 0.4) \) \( w_{14} = (0.2, 0.6, 0.2) \) \( w_{15} = (0.4, 0.2, 0.4) \) \( w_{16} = (0.4, 0.4, 0.2) \) \( w_{17} = (0.6, 0.2, 0.2) \)
Weights: \( w_1=<0,0,1> \) \( w_2=<0,1,0> \) \( w_3=<1,0,0> \) \( w_4=<0,.2,.8> \) \( w_5=<0,.4,.6> \) \( w_6=<0,.6,.4> \) \( w_7=<0,.8,.2> \) \( w_8=<.2,0,.8> \) \( w_9=<.4,0,.6> \) \( w_{10}=<.6,0,.4> \) \( w_{11}=<.8,0,.2> \) \( w_{12}=<.2,.8,0> \) \( w_{13}=<.4,6,0> \) \( w_{14}=<.6,4,0> \) \( w_{15}=<.8,2,0> \) \( w_{16}=<.2,2,6> \) \( w_{17}=<.2,4,4> \) \( w_{18}=<.2,6,2> \) \( w_{19}=<.4,2,4> \) \( w_{20}=<.4,4,2> \) \( w_{21}=<.6,2,2> \)
Cache size vs avg speed up for mid10

- **Small size**
- **Medium size**
- **Large size**

![Graph showing speed up for different cache sizes](image)
Average views in cache (low 20, no k)

weights

avg views in cache

Workload 0
Workload 1
Workload 2
Workload 3
Workload 4
Workload 5
Dynamic change rules

• If weights vary across all workloads, how to do we detect a workload and adjust the weight?

• A good formula should have a wide range of values and predict performance
  - ComputationCost = serverCost + transferCost
  - Change in the range of values may or may not be significant.
Relationship between change in std dev

![Graph showing relationship between workloads and correlation with speedup.](image)

- F1 vs Speedup
- F2 vs Speed up
- F3 vs Speed up
An Example - Mapping

- Views = maps of specific types
- Reuse if a map can be cropped to satisfy the current request
- Support for map operations → Panning
  - If the user has requested a map, it is very likely that she will pan once in that map
  - If the user panned to left, it is more likely that she will pan to the same direction than the opposite direction

[Learn and insert an actual model of user behavior here.]
An Example - Mapping

- New statistic: panFactor: how many pans are available to the last direction the user moved
- Order formula:
  - FOCUS: panFactor * last_use/now
  - Use a weighted sum
    - \( w_1 \times \text{FOCUS} + w_2 \times F_1 \ldots \)
An Example - Mapping

- It is likely that the user is going to perform a zoom-in or a zoom-out operation on the same area next!
  - Prefetch what is likely to be useful in the future
  - View reuse method generates a set of “likely” queries Q1, ..., Qk
  - Use a prefetch order formula to determine the priority of each Qi
  - Insert all Qi to a Prefetch Queue (PfQ)
Déjà vu?

- Prefetch queue operates exactly like priority queue
  - Available space (PQ) \(\rightarrow\) available bandwidth (PfQ) [prefetch only when idle]
  - How to allocate bandwidth? \(\rightarrow\) call the highest priority views
  - Optimize \(\rightarrow\) Savings in the overall expected delay
  - How to define priority?
    - Focus based on likelihood
    - Complexity \(\rightarrow\) based on map complexity
An Example - Mapping

• Highly complex views will save most in terms of delay, but will keep the server busy

• What size to get?
  - A larger view is more likely to be useful now and in the future
  - A smaller view will incur smaller overall cost in case of a mistake

• What entry criteria to use?
  - Theoretically, a generated request is less likely to be repeated than an actual (human generated) request
Conclusions

• CAVES provides a general purpose storage management system that is programmable and adaptive.
• Research funded by NSF
• Group members
  S. Adali
  C. Carothers
  D. Spooner
  G. Yaun
  C. Chiang
Research topics

Determining semantic overlap of views is too costly in practice
- Approximate measures
- Computed by view reuse methods incrementally
- Extend the PQ management function to include a dependency graph and cost measures

Semantic caching [Franklin, Godfrey]
Data Cube optimizations [Harinayan, Ullman]
Research Topics

Use simulation as a decision module using actual load characterizations to find
- Optimal/near optimal weight distributions
- Useful measures that predict performance
- Discover useful weight change rules

Probabilistic models for network loads
Tests with actual data sets
Research topics

• Scalability: networked or hierarchical storage servers [Kalnis Papadias]
  - Each server has its own priority queue and priority functions
  - Objectives vary at each level
• The priority of a view depends on the priority of views in other servers
• Each server observes a different request distribution, aggregated at higher levels