Dynamic Data Dissemination (Publish/Subscribe)

Dynamic Data
- Traffic data: packets thru switches / vehicles on highways
- Stock prices, Sport Scores
  - rapid and unpredictable changes
  - time critical, value critical
  - used in on-line monitoring, decision making

More and more of data gathered from the web/internet is dynamic.

Coherency of Dynamic Data
- **Strong coherency**
  - The client and source always in sync \( U(t) = S(t) \)
  - Strong coherency is expensive!
- **Relax strong coherency**: \( \Delta \) - coherency
  - **Time domain**: \( \Delta t \) - coherency
  - **Value domain**: \( \Delta v \) - coherency
    - The difference in the data values at the client and the source bounded by \( \Delta v \) at all times
    - E.g.: temperature changes greater than 1 degree

\[ \forall t, |U(t) - S(t)| < \Delta v \]

Modes of Data Dissemination
- **Pull**:
  - Client pulls data from the source.
- **Push**:
  - Source pushes data of interest to the client.

Generic Architecture

Data sources
End-hosts
Wired hosts
Mobile hosts
Network
Servers
Proxies/caches
Data sources
End-hosts
Wired host
Mobile host
Network
Servers
**Where should the queries execute?**

- At clients
  - Can’t optimize across clients, links
- At source (where changes take place)
  - Advantages
    - Minimum number of refresh messages, high fidelity
  - Main challenge
    - Scalability
    - Multiple sources hard to handle
- At Data Aggregators -- DAs/proxies -- placed at edge of network
  - Advantages
    - Allows scalability through consolidation, multiple data sources
  - Main challenge
    - Need mechanisms for maintaining data consistency at DAs

**The Basic Problem...**

- To create a scalable content dissemination network (CDN) for streaming/dynamic data.

  **Metric:**
  
  Fidelity: 
  % of time coherency requirement is met

  **Goal:** To achieve high fidelity and resiliency

**Cooperative Repository Architecture**

- Clients request for different data items by specifying coherence requirements for each item
- Repositories derive their requirements from the client requirements
- Source pushes the changes of interest to repositories
- Repositories cooperate with each other and the source to serve clients

**Dissemination Graph: Example**

Data Set: p.q.r  Max # push connections: 2

**Challenges**

- Given the data and coherency needs of repositories, how should repositories cooperate to satisfy these needs?
- How should repositories refresh the data such that coherence requirements of dependents are satisfied?
- How to make repository network resilient to failures?
- Given the data and the coherency available at repositories, how to assign clients to the repositories?
Data Dissemination (Supp reading 1)

- Different users have different coherency req for the same data item.
- Coherency requirement at a repository should be at least as stringent as that of the dependents.
- Repositories disseminate only changes of interest.

Data Dissemination

A repository $P$ sends changes of interest to the dependent $Q$ if

$$|x^P - x^Q| \geq c^Q$$

$x^P$: value of $x$ at $P$
$c^Q$: coherency req of $x$ at $Q$

Dissemination Algorithms

- Source Based (Centralized)
- Repository Based (Distributed)

Source Based Dissemination Algorithm

- For each data item, source maintains
  - unique coherency requirements of repositories
  - the last update sent for that coherency
- For every change,
  - source finds the maximum coherency for which it must be disseminated
  - tags the change with that coherency
  - disseminates (changed data, tag)
Repository Based Dissemination Algorithm

A repository $P$ sends changes of interest to the dependent $Q$ if

$$|x^P - x^Q| \geq c^Q - c^P$$

Dissemination Algorithms

- Repository based algorithm requires fewer checks at source
- Source based algorithm requires less messages

Logical Structure of the CDN

- Repositories want many data items, each with different coherency requirements.
- Issues: Which repository should serve what to whom?

Constructing the Layout Network

Algorithm LeLA (Level by Level Algorithm):
Insert repositories one by one
- Check level by level starting from the source
  - Each level has a load controller.
  - The load controller tries to find data providers for the new repository($Q$).

Selecting Data Providers

- Repositories with low preference factor are considered as potential data providers.
- The most preferred repository with a needed data item is made the provider of that data item.
- The most preferred repository is made to provide the remaining data items (some of these may not be currently disseminated via the node)
Preference Factor

- Resource Availability factor:
  Can repository (P) be the provider for one more dependent?
- Data Availability Factor:
  #data items that P can provide for the new repository Q.
- Computational delay factor:
  #dependents P provides for.
- Communication delay factor:
  network delay between the 2 repositories.

\[
\text{Comm delay (P, Q) \times \text{number dependents(P)}}
\]
\[
\frac{\text{Number of data items P can serve Q}}{\text{P can provide for the new repository Q.}}
\]

Loss of fidelity for different coherency requirements

The less stringent the coherency requirement, the better the fidelity.
For little/no cooperation, loss in fidelity is high.
Too much cooperation?

Controlled cooperation

Actual degree of cooperation

\[
= \frac{\text{average network delay}}{\text{average comp delay} \times \text{#interested dependents}}
\]

Subject to offered degree of cooperation

Controlled cooperation is essential

If parents are not chosen judiciously

It may result in
- Uneven distribution of load on repositories.
- Increase in the number of messages in the system.

Increase in loss in fidelity!

But ...

Loss in fidelity increases for large # data items.

Repositories with stringent coherence requirements should be closer to the source.
**Data Item at a Time Algorithm**

- A dissemination tree for each data item.
- Source serving the data item is the root.
- Repositories with more stringent coherency requirements are placed closer to the root.

**DiTA**

- Repository $N$ needs data item $x$.
- If the source has available push connections, or the source is the only node in the dissemination tree for $x$.
  - $N$ is made the child of the source.
- Else.
  - Repository is inserted in most suitable subtree where
    - $N$'s ancestors have more stringent coherency requirements
    - $N$ is closest to the root

**Most Suitable Subtree?**

- $l$: smallest level in the subtree with coherency requirement less stringent than $N$'s.
- $d$: communication delay from the root of the subtree to $N$.
- Smallest ($l \times d$): most suitable subtree.

Essentially, minimize communication delays!

**Example**

Initially the network consists of the source.

A and $B$ request service of $q$ with coherency requirement $0.2$.

What if $C$ requests service of $q$ with coherency requirement of $0.1$?

**Example**

Initially the network consists of the source.

A and $B$ request service of $q$ with coherency requirement $0.2$.

What if $C$ requests service of $q$ with coherency requirement of $0.1$?

$C$ requests service of $q$ with coherency requirement $0.1$. 
Comparing LeLA & DiTA

Handling Failures in the Network

• Need to detect permanent/transient failures in the network and to recover from them
• Resiliency is obtained by adding redundancy
• Without redundancy, failures lead to loss in fidelity
• Adding redundancy can increase cost but it is possible to recover from loss of fidelity
• Handle failures such that the cost of adding resiliency is low!

Passive/Active Failure Handling

• Passive failure detection:
  – Parent sends I'm alive messages at the end of every time interval.
  – What should the time interval be?
• Active failure handling:
  – Always be prepared for failures.
  – For example: 2 repositories can serve the same data item at the same coherency to a child.
  – This means lots of work but it ensures greater loss in fidelity.
  
  Need to be clever!

Middle Path

Let repository $R$ want data item $x$ with coherency $c$.

A backup parent $B$ is found for each data item that the repository needs.

At what coherency should $B$ serve $R$?

$B$ serves $R$ with coherency $k \times c$

If a parent fails

• Detection: Child gets two consecutive updates from the backup parent with no updates from the parent
• Recovery: Backup parent is asked to serve at coherency $c$ until we get an update from the parent

Adding Resiliency to DiTA

• A sibling of $P$ is chosen as the backup parent of $R$
• If $P$ fails, $A$ serves $B$ with coherency $c$ and change is local.
• If $P$ has no siblings, a sibling of nearest ancestor is chosen. Else the source is made the backup parent.

Cost-Based Approach (Supp reading 2)

- Exiting heuristics (e.g., DiTA):
  - The parent node should have a more stringent coherency requirement than its children;
  - Impose an apriori fanout constraint on each node;
- Potential problems:
  - A slow node with a very stringent requirement is put at the top of the tree
  - Multiple rounds of trial and error to obtain a good fanout constraint
  - Cannot adapt to changes of the system
    - E.g., coherency requirements, workload in the nodes, transfer delays etc.

Cost-based Solution

- A cost-based approach:
  - Can explore a larger solution space
  - No trial and error is needed
- Assumptions?
- Adapting the dissemination tree at run time

Cost model

- \( LF_i = r_i \cdot D_i \)
  - \( r_i \): the avg update arrival rate for the \( i \)th node
  - \( D_i \): the avg delay of each update message for the \( i \)th node
- \( D_i \) includes the aggregated
  - comm delay
  - queueing time (estimated using M/M/1 queueing model)
  - processing time
  - in the path from source S to the \( i \)th node;

Local Transformation Rules

- Node Promotion

- Node Demotion

- Parent-Child Swap
Local Transformation Rules

• Cousin Swap

\[ \text{\ldots} \]

\[ k \]

\[ g \]

\[ h \]

\[ i \]

\[ j \]

\[ \ldots \]

\[ \ldots \]

\[ \ldots \]

Local Transformation Rules

• Nephew Adoption

\[ \text{\ldots} \]

\[ k \]

\[ g \]

\[ h \]

\[ i \]

\[ j \]

\[ \ldots \]

\[ \ldots \]

\[ \ldots \]

Local Transformation Rules

• Uncle-Nephew Swap

\[ \text{\ldots} \]

\[ k \]

\[ g \]

\[ h \]

\[ i \]

\[ j \]

\[ \ldots \]

\[ \ldots \]

\[ \ldots \]

Benefit Estimation

• Re-computing the new cost from scratch incurs large overhead
• Fortunately, each transformation affects only part of the tree
  – We can compute the \( \Delta \) cost in \textit{constant time} using only local information.

Benefit Estimation - Example

• Node Promotion

\[ \text{\ldots} \]

\[ \text{Affected area} \]

\[ i \]

\[ j \]

\[ \ldots \]

\[ \ldots \]

\[ \ldots \]

Making adaptation decisions

• Centralized approach
  – Less scalable and reliable
• Fully distributed approach
  – Conflicts
    • Two nodes make contradicting decisions
  – Resource wastage
    • Two nodes arrive in the same decision
Making adaptation decisions

- A token-based distributed approach:
  - Each node only considers the transformations involving its children and grandchildren;
  - A node can make decisions only when it holds a token;
  - At the start of each round, a token is generated by the root;
  - The token is passed to the children once the adaptation is done;
  - Each node chooses the transformation that has the highest benefit.

Static Construction Algorithms

- For static environment or initial tree construction
- Heuristic Algorithm
  - Sort the nodes in ascending order of comm. delays and processing times;
  - For each node in the list:
    - Select a position in the tree so that the average LF of the whole tree is minimized;
- Simulated Annealing
  - Use the above local transformation rules;

Sensitivity to processing time

Dynamic Environment

Conclusion

- Dynamic data has to be disseminated to maximize fidelity
- Basic approach is to design a dissemination tree/graph
- Both heuristics and cost-based approach have been presented
- Cost-based approach is shown to be superior