HY559
Infrastructure Technologies for Large-Scale Service-Oriented Systems

Kostas Magoutis
magoutis@ics.forth.gr
http://www.ics.forth.gr/~magoutis
Advantages of clusters

• Scalability

• High availability

• Commodity building blocks
Challenges of cluster computing

• Administration

• Component vs. system replication

• Partial failures

• Shared state
ACID semantics

• Atomicity

• Consistency

• Isolation

• Durability
BASE semantics

• Stale data temporarily tolerated
  – E.g., DNS

• Soft state exploited to improve performance
  – Regenerated at expense of CPU or I/O

• Approximate answers delivered quickly may be more valuable than exact answers delivered slowly
Architecture of generic SNS
Three layers of functionality

<table>
<thead>
<tr>
<th><strong>Service:</strong></th>
<th>Service-specific code</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Workers that present human interface to what TACC modules do, including device-specific presentation</td>
<td></td>
</tr>
<tr>
<td>• User interface to control the service</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>TACC:</strong> Transformation, Aggregation, Caching, Customization</th>
</tr>
</thead>
<tbody>
<tr>
<td>• API for composition of stateless data transformation and content aggregation modules</td>
</tr>
<tr>
<td>• Uniform caching of original, post-aggregation and post-transformation data</td>
</tr>
<tr>
<td>• Transparent access to Customization database</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>SNS:</strong> Scalable Network Service support</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Incremental and absolute scalability</td>
</tr>
<tr>
<td>• Worker load balancing and overflow management</td>
</tr>
<tr>
<td>• Front-end availability, fault tolerance mechanisms</td>
</tr>
<tr>
<td>• System monitoring and logging</td>
</tr>
</tbody>
</table>
A reusable SNS support layer - scalability

- Replicate components of SNS architecture for fault tolerance, high availability, and scalability

- Shared non-replicated system components do not become bottleneck
  - Network, resource manager, user-profile database

- Simplify workers by moving functionality to front-end
  - Manage network state for outstanding requests
  - Service-specific worker dispatch logic
  - Access profile database
  - Notify user in service-specific way when a worker fails
Load balancing

• Manager tasks
  – Collect load information from workers
  – Synthesize load balancing hints based on policy
  – Periodically transmit hints to front ends
  – Load balancing and overflow polices left to operator

• Centralized vs. distributed design
Overflow growth provisioning

• Internet services exhibit bursts of high load (the "flash crowds")

• Overflow pool can absorb such bursts
  – Overflow machines are not dedicated to service
Soft state for fault tolerance, availability

• SNS components monitor one another using *process peer* fault tolerance
  – When component fails, a peer restarts it on another machine
  – Cached stale state carries surviving components through failure
  – Restarted component gradually rebuilds soft state

• Use timeouts as additional fault-tolerance mechanism
  – If possible to resolve, perform necessary actions
  – Otherwise, service layer decides how to proceed
TACC programming model

• Transformation
  – An operation that changes the content of a data object
  – E.g., filter, re-render, encrypt, compress

• Aggregate
  – Collect data from several objects and collate it in a pre-specified way

• Cache
  – Store post-transformation or post-aggregation content in addition to caching original Internet content

• Customize
  – Track users and keep profile information (in ACID database), deliver information automatically to workers
TranSend - front-end

• Front-end presents HTTP interface to clients

• Request processing includes
  – Fetching Web data from cache (or Internet)
  – Pairing up request with user’s customization preferences
  – Send request, preferences to pipeline of distillers
  – Return result to client
Load balancing manager

- Client-side JavaScript balances load across front-ends
- Centralized load balancer
  - Tracks location of distillers
  - Spawns new distillers on demand
  - Balances load across distillers of same class
  - Provides fault-tolerance and system tuning
- Manager beacons existence on IP multicast group
- Workers send load information through stubs
- Manager aggregates load info, computes averages, piggybacks to beacons to manager stubs
Fault-tolerance

• Manager, distillers, front-ends are process peers
  - Process peer functionality encapsulated in manager stubs

• Ways to detect failure
  - Broken connections
  - Timeouts
  - Loss of beacons

• Soft state simplifies crash recovery
User profile database

• Allows registering user preferences
  – HTML forms or Java/JavaScript combination applet

• Implemented using gdbm (Berkeley DB)
  – Read cache at the front-ends
Cache nodes

• Harvest object cache on four nodes

• Deficiencies
  – All sibling caches queried on all requests
  – Data cannot be injected into it
  – Separate TCP connection per HTTP request

• Fixes
  – Hash key space across caches and rebalance (mgr stub)
  – Allow injection of post-processed data (worker stub)
Datatype-specific distillers

• Distillers are workers that perform transformation and aggregation

• Three parameterizable distillers
  – Scaling and low-pass filtering of JPEG images
  – GIF to JPEG conversion followed by JPEG degradation
  – Perl HTML transformer
How TranSend exploits BASE

- Stale load-balancing data
- Soft state
- Approximate answers
HotBot implementation

• Load balancing
  – Workers statically partition search-engine database
  – Each worker gets share proportional to its power
  – Every query goes to all workers in parallel

• Failure management
  – HotBot workers are not interchangeable since each worker uses local disk
  – Use RAID to handle disk failures
  – Fast restart minimizes impact of node failures
  – Loss of 1/26 machines takes out 3M/54M documents
# TranSend vs. HotBot

<table>
<thead>
<tr>
<th>Component</th>
<th>TranSend</th>
<th>HotBot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load balancing</td>
<td>Dynamic, by queue lengths at worker nodes</td>
<td>Static partitioning of read-only data</td>
</tr>
<tr>
<td>Application layer</td>
<td>Composable TACC workers</td>
<td>Fixed search service application</td>
</tr>
<tr>
<td>Service layer</td>
<td>Worker dispatch logic, HTML / JavaScript UI</td>
<td>Dynamic HTML generation, HTML UI</td>
</tr>
<tr>
<td>Failure management</td>
<td>Centralized but fault-tolerant using process-peers</td>
<td>Distributed to each node</td>
</tr>
<tr>
<td>Worker placement</td>
<td>FE’s and caches bound to their nodes</td>
<td>All workers bound to their nodes</td>
</tr>
<tr>
<td>User profile (ACID) database</td>
<td>Berkeley DB with read caches</td>
<td>Parallel Informix server</td>
</tr>
<tr>
<td>Caching</td>
<td>Harvest caches store pre- and post-transformation Web data</td>
<td>integrated cache of recent searches, for incremental delivery</td>
</tr>
</tbody>
</table>