Pragmatic Clustering

Mike Cannon-Brookes
CEO, Atlassian Software Systems

Confluence

- Largest enterprise wiki in the world
- 2000 customers in 60 countries
- J2EE application, ~500k LOC
  - Hibernate, Lucene, Spring, WebWork
- Deployed by customers behind the firewall

http://www.atlassian.com/confluence

Why Cluster?

- Horizontal scalability
  - More machines = more scale
  - Customer: our use of Confluence is increasing beyond a single machine.
- High availability
  - More machines = more *overall* uptime
  - Customer: our wiki cannot be down.

The Plan

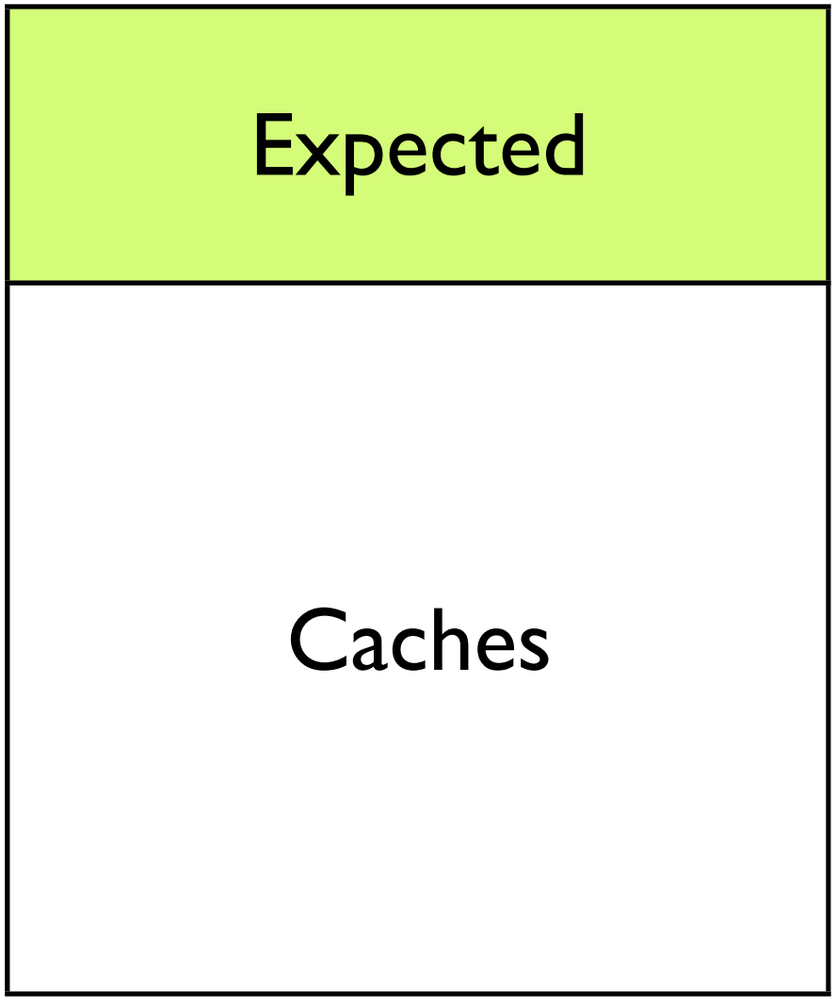
Start by asking a simple question:

“What breaks when 2 Confluence instances run against a single database?”

Breakages? For example dirty cache reads.

The Answer

The Answer



The Answer

Expected	Actual
Caches	Caches Lucene Indexes Events Configuration File Access Scheduled Jobs

Strategies

Strategy	Central Share	Distributed Share	Synchronize	Temporize
Example	Database	Clustered cache	Lucene index per node	Thumbnails

With synchronised, need to handle desynchronisation case (ie new node coming up)

Clustered cache is ‘distributed’ share, database is ‘centralised’ share

First...
Get used to trade offs!

There is no perfect design. Pick the closest to you and handle the trade offs.

Our Design

- All nodes identical
- No shared file system
- Sticky sessions
- Low admin overhead
- Assume the database scales

No master / slave, as this simplifies the topology.
Confluence is an end user application, we can't assume the customer has a shared FS
No session replication overhead is great, small downside of session loss.
We ship this to end customers, so administration must be simple. Example here is the cluster name vs IP/port combination.
Note, this design does not please _everyone_ - eg www.ibm.com remote clusters don't work.

Implementation Lessons

Caches, Cluster Management, Lucene, Events, Files, Scheduled Jobs, Testing

1. Caches

Problem

- Cached data per node leads to dirty reads
- No caches = most scalable!
  - Push all work into the database
  - Not performant for most apps
- Otherwise, need a replicated cache

1. Caches

Solution

Strategy: *distributed share*

- Use a library
  - Commercial: Tangosol Coherence, Gigaspaces, Terracotta, Gemstone
  - Open Source: Ehcache, OSCache, JBoss Cache
- Confluence: Tangosol Coherence
- Make everything serializable
- Select partitioned vs replicated caches

Amusing anecdote – first time we brought up a clustered cache, the entire development team became a cluster!

2. Cluster Management

Problem

- Node discovery & heartbeat
- Cluster information
- Execute code 'across' the cluster

Solution

1. Choose a good library
2. Implement it all yourself!

Amusing anecdote - first time we brought up a clustered cache, the entire development team became a cluster!

2. Cluster Management

Solution

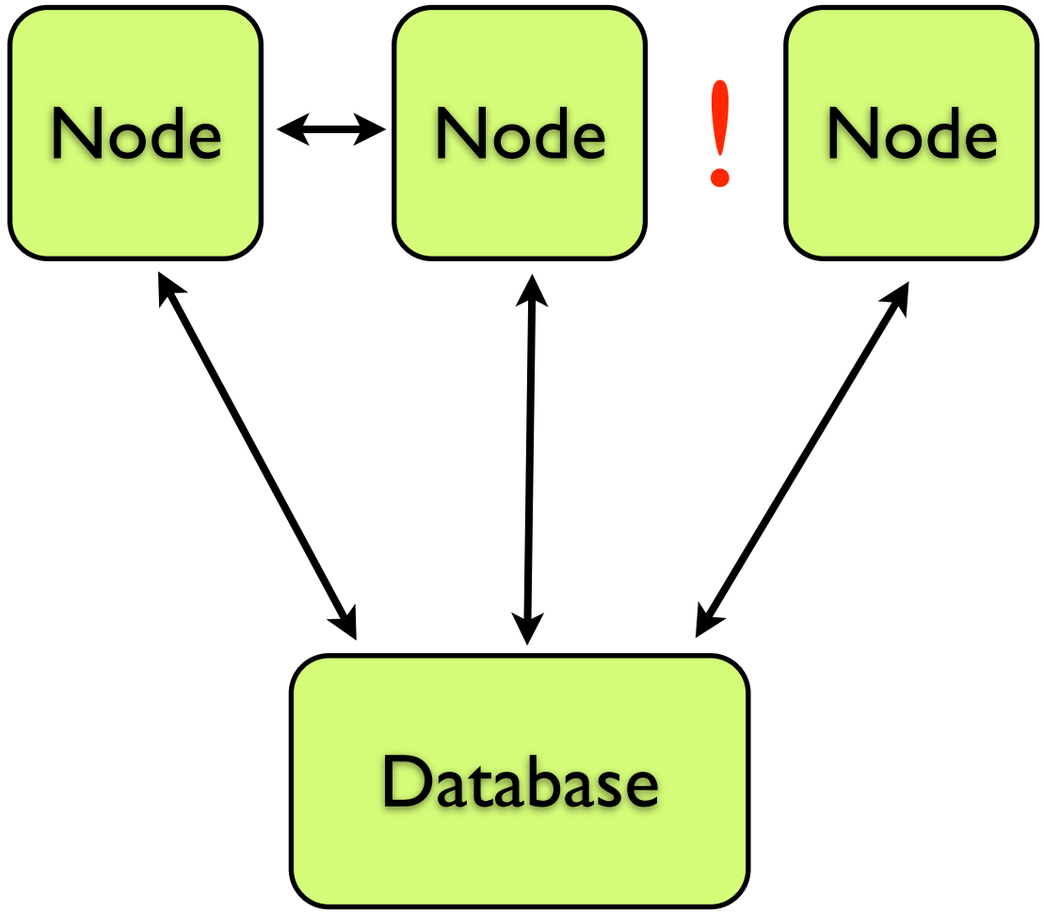
- Low admin overhead is *hard*
  - Cluster name → multicast IP / port
    - eg "XYZ Cluster" → 234.3.1.2 : 3535
  - Allow complex config options via XML
- Beware automated discovery!
  - eg entire dev team *working together*

Amusing anecdote – first time we brought up a clustered cache, the entire development team became a cluster!

2½. Cluster Split

Problem

- Cluster 'splits' into fragments
- Each fragment can see DB, but not others
- Potential for massive data corruption



Extranet- firewall blocked multicast between machines, but TCPIP traffic to database was fine.

2½. Cluster Split

Solution

- Detect split and shutdown
  - Periodically write a random number to DB and cache
    - Also when nodes join / leave cluster
  - Periodically check that number has not changed
  - If number is detected to have changed, split!
    - Shutdown fragment to all access.
    - Write new number, so others shutdown next check.

3. Lucene

Problem

- Indexing vital to all transactional applications
- Must live on file system
- Doesn't like network file systems
  - Too slow, writes must be serialized

3. Lucene

Solutions

1. Dedicated search nodes
    - No longer all nodes identical
    - Have to manage two node ‘classes’

3. Lucene

Solutions

1. ~~Dedicated search nodes~~
2. One node indexes
   - Batch push of index diffs around cluster
   - Need master/slave - not identical
   - Potential for data loss if index node goes down
   - Index diff sending fragile & complex

3. Lucene

Solutions

Strategy: *synchronize*

1. ~~Dedicated search nodes~~
2. ~~One node indexes~~
3. All nodes index
   - Persistent queue of index operations (in database, no JMS)
   - 2 hour queue (downtime window)
   - If down longer, re-index on join
     - Remember - index is derived data
   - Trade off - work is being done everywhere

4. Events

Problem

- Confluence largely event driven internally
- Should events be propagated around cluster?
- Event scopes:
  - per node - work done on *originating* node
  - per cluster - work done on *every* node

4. Events

Solution

1. Local listener hears a cluster scoped event
   - eg PluginInstallEvent implements ClusterableEvent
2. Wraps in new ClusterBroadcastEvent
3. Broadcasts CBE around cluster
   - Unreliable messaging (via Tangosol)
   - Offline nodes don't care about events
4. Listeners then listen for ClusterBroadcastEvent
   - Unwrap, check contents, act if necessary (ie plugin install)

4. Events

Solution

- Beware exponential scale problems!
- Avoid clustered events where possible
  - Use local event + clustered cache
  - Reduce ‘work done everywhere’
  - Sometimes not possible - ie classloading

5. Files

Problem

- Efficient to store files on disk
- Local files not visible to cluster
- Confluence:
  - Configuration files
  - Attachments
  - Installed plugins
  - Thumbnails
  - Files generated for download (eg charts & PDFs)

5. Files

Solutions

Strategy: *central share*

1. Network file system / SAN
   - Never as fast as local file system
   - File locking problems
   - Confluence: can't guarantee user has one!

5. Files

Solutions Strategy: *central share & temporize*

1. ~~Network file system / SAN~~
2. Use database, remaining files are temporary
   - Config files & attachments moved into database
     - Attachments spooled to local file system to serve
   - Thumbnails, PDFs, charts generated on each node
     - Simpler architecture outweighs duplicated work
     - Trash periodically, regenerate anytime

Example of Telenor problem with Xmas party photos and transactions in database bringing Confluence down

6. Scheduled Jobs

Problem

- Confluence uses Quartz for scheduled jobs
- Different job types:
  - delete local temp files - on every node
  - send notification email - once per cluster

6. Scheduled Jobs

Solution — Strategy: *distributed share*

- Divide up jobs into two scopes
  - LocalJob - executed on every node
  - ClusterJob - executed once per cluster
- Per node - jobs behave as normal
- Per cluster - ClusterJobStore
  - Implemented new Quartz JobStore
  - Backed by clustered cache

7. Testing

Problem

- Automated, repeatable testing is crucial
  - Automated testing of clusters is hard!
- Confluence:
  - Uses Bamboo for continuous integration

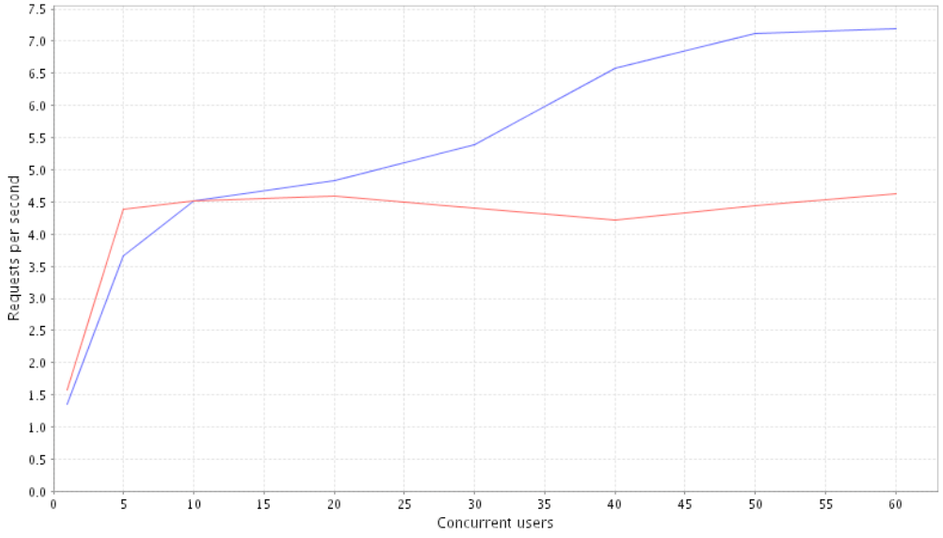
7. Testing

Solution

- Functional Testing
  - Problems similar to multithreaded testing
  - Confluence:
    - JWebUnit with switched WebConversations
    - Cargo to fire multiple web containers

7. Testing

Solution



- Load & Performance Testing
  - Repeatable load test
  - Measure continual progress
  - Confluence:
    - Originally reused JWebUnit tests
    - Switched to raw HTTP, less overhead

Q&A

Java guru? J2EE junkie? Think VB is for drinking?

Join the best Java engineering team at
Australia's fastest growing software company!

http://www.atlassian.com/about/jobs
Email me: mike@atlassian.com