Scaling OSNs without pains

Josep M. Pujol, Georgos Siganos, Vijay Erramilli and Pablo Rodriguez

Telefonica Research, Barcelona
Motivation: avoiding pain

• Assumption
  Everything is easier when all data is local

• Why?
  – Many systems start with this implicit assumption
  – Easy to implement, debug and manage
  – Distributed programming is not for everyone
  – Programming for a single box let you focus on functionality

• The source of pain is scalability
  – A fictional one-box Twitter implementation would have tables growing 150M rows/day, serving 1000’s requests/s
Goal: automatic scaling

• Scaling...
  – Scaling up the front-end is “easy” with the Cloud
    • servers are independent, and therefore, interchangeable
  – The back-end is a different story
    • servers are no longer independent

• Is it possible to scale an application that...
  – it is designed to run centralized (all data local)
  – with a middleware transparent to the application
  – and that is application agnostic  ?
Goal: automatic scaling

• Scaling...
  – Scaling up the front-end is “easy” with the Cloud
    • servers are independent, and therefore, interchangeable
  – The back-end is a different story
    • servers are no longer independent

• Is it possible to scale an application that...
  – it is designed to run centralized (all data local)
  – with a middleware transparent to the application
  – and that is application agnostic ?

• It depends...
Scaling up...

• Full replication
  – Maintains the data locality assumption
  – Load balancing across $N$ perfect copies
  – Server requirements does not decrease with $N$

• Horizontal partitioning or *sharding*
  – Data is split evenly in $N$ servers, usually by user
  – Server requirements decreases linearly with $N$
  – Maintains the data locality assumption
    • when the splits are disjoint (independent)

• *bad news for OSNs!!*
The hair-ball problem

• Splits in OSN can never be disjoint because:
  – Operations on user $i$ require access to the data of other users
  – at least one-hop away.

• From graph theory:
  – there is no partition that for all nodes all neighbors and itself are in the same partition if there is a single connected component.
  – Percolation occurs at $<k>=1$ in ER graphs.
How do OSNs deal with this...

• Just drop the data locality assumption...
  – When the data is on RDBMS
    • Selects and joins across multiple images of the database
    • Too inefficient even for de-normalized schemas
  – When data is on Key-Value Stores
    • Efficient (when data is de-normalized)
    • But not pain-free by far
      » lose SQL query language, programmatic queries
      » mix the data model with the application logic
      » not as robust as SQL (yet)
      » suffer from high traffic volume
Our Solution: Partition + Replication

1) PARTITION: assign a master server to each user.

\[ P : N \rightarrow S \]

2) REPLICATION: create replicas for bridges

\[ \exists v \in N_u : P(u) \neq P(v) \]

so all data is local

Replication is done at the row level

>> The ratio between masters and replicates is the overhead
The relevance of the Partition algorithm

• Random partition (sharding)
  – $N/S$ master users in each server
  – However the replicas in each server is
    $\left( N - \frac{N}{S} \right) \left( 1 - e^{\frac{\langle k \rangle}{S} - \langle k \rangle} \right)$
  – Becomes full replication unless $<k>$ very small
  – The replication overhead $r_u$ is practically $S$
  – User replication can be done at the data-center level to reduce overhead as a partial solution
    – More efficient joins, but it is still distributed, data non-local

• Random partition fails to leverage the underlying community structure of social networks.
The relevance of the Partition algorithm

- **Social Network partition:**
  - *Rationale:* replicas are not user-dependent but community dependent. Thus if you group tight-knit communities together the number of required replicas is minimized.
  - We used our clustering algorithm MO+
    - based on the BGLL modularity optimization algorithm
    - modified to give equal size partitions
    - spectral partitioning METIS was also evaluated
  - No analytic solution for the number of replicas. It is a function of the community structure and $<k>$
    - For 2.4M users of Twitter in 32 machines, $r_u = 2.45$
Results: Empirical Evaluation

Replication overhead for Twitter and Orkut datasets

Replication overhead growth sub-linearly with the number of servers.

For 512 servers, each server would have an average replication factor (overhead) of 5.8 for Twitter and 18.5 for Orkut.
Results: Empirical Evaluation

Server requirements per number of servers

Server requirements decrease as $r_u/S$, which is a potential law regardless of the replication overhead $r_u$.

We can project scaling costs if we know the exponent of the power law as $S^{-\gamma}$. Twitter is $S^{1.13}$

>> Give us a single box app that supports 50K users and, it will scale to 100M users using 5300 servers...

... or to 1M users using 30 servers
Extending SN Partition

• SN partitioning has one important limitation:
  – It minimizes the number of inter-partition edges across partitions for a given user, but it is agnostic to the number of partitions where a bridge exists

![Diagram showing number of edges and replication]

• Partitioning must be an online process:
  – In order to deal with user growth and densification
Combining replication overheads

Replication is common place for redundancy and HA:

- at least $k$ copies of the user’s data imposes a replication overhead of $r_k = k-1$.

The replication for scalability ($r_s$) can be combined to the replication for redundancy so

$$r_{k+s} < r_k + r_s.$$
Testing the system in the *wild*

- The most important ongoing work is the empirical evaluation of the system
  - We use an open-source Twitter-clone application (PHP+MySQL)
  - Apply our middleware on top of MySQL
  - Stress test the Twitter clone with our middleware up to the current traffic level of Twitter.

- **Aiming to**
  - show that it is possible to scale up OSN’s
    - without leaving the centralized paradigm of data locality
    - with a middleware that is application agnostic

- **A couple of notes...**
System Architecture Diagram

Reverse Proxy
Lookup Table $H(data_{id})$

OHR:
Let reads through
Capture writes:
- transform $data_{id}$ to $app_{id}$
- look up table for replicas $H'(app_{id})$
- replay the write to other DB's

FE  DB  server

OHR

FE  DB  server

N/S Masters
$r_u$ N/S Replicates

Eventual consistency

NetDB – SOSP 2009
Thanks a lot!

Josep M. Pujol @ Telefonica Research
http://research.tid.es/internet/