The Little Engine(s) that could: Scaling Online Social Networks

Josep M. Pujol, Vijay Erramilli, Georgos Siganos, Xiaoyuan Yang
Nikos Laoutaris, Parminder Chhabra and Pablo Rodriguez

Telefonica Research, Barcelona
Problem

Solution

Implementation

Conclusions
Scalability is a pain: the designers’ dilemma

"Scalability is a desirable property of a system, a network, or a process, which indicates its ability to either handle growing amounts of work in a graceful manner or to be readily enlarged"

The designers’ dilemma:

wasted complexity and long time to market

vs.

short time to market but risk of death by success
Towards transparent scalability

The advent of the Cloud: virtualized machinery + gigabit network

Hardware scalability ✓

Application scalability ×
Towards transparent scalability

Elastic resource allocation for the Presentation and the Logic layer:
- More machines when load increases.
- Components are stateless, therefore, independent and duplicable.

Components are interdependent, therefore, non-duplicable on demand.
Scaling the data backend...

• Full replication
  – Load of read requests decrease with the number of servers
  – State does not decrease with the number of servers
Scaling the data backend...

• Full replication
  – Load of read requests decrease with the number of servers
  – State does not decrease with the number of servers
  – Maintains data locality
Scaling the data backend...

• **Full replication**
  – Load of read requests decrease with the number of servers
  – State does not decrease with the number of servers
  – **Maintains data locality**

Operations on the data (e.g. a query) can be resolved in a single server from the application perspective
Scaling the data backend...

• **Full replication**
  – Load of read requests decrease with the number of servers
  – State does not decrease with the number of servers
  – Maintains data locality

• **Horizontal partitioning or sharding**
  – Load of read requests decrease with the number of servers
  – State decrease with the number of servers
  – Maintains data locality as long as...
Scaling the data backend...

• **Full replication**
  – Load of read requests decrease with the number of servers
  – State does not decrease with the number of servers
  – **Maintains data locality**

• **Horizontal partitioning or sharding**
  – Load of read requests decrease with the number of servers
  – State decrease with the number of servers
  – **Maintains data locality** as long as...

... the splits are disjoint (independent)

**Bad news for OSNs!!**
Why sharding is not enough for OSN?

• Shards 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.

Data locality cannot be maintained by partitioning a social network!!
A very active area...

• Online Social Networks are massive systems spanning hundreds of machines in multiple datacenters
  • **Partition and placement to minimize network traffic and delay**
    – Karagiannis et al. “Hermes: Clustering Users in Large-Scale Email Services” -- SoCC 2010
    – Agarwal et al. “Volley: Automated Data Placement of Geo-Distributed Cloud Services” -- NSDI 2010
  • **Partition and replication towards scalability**
    – Pujol et al “Scaling Online Social Networks without Pains” -- NetDB / SOSP 2009
Problem
Solution
Implementation
Conclusions
OSN’s operations 101
OSN’s operations 101
OSN’s operations 101
OSN’s operations 101
OSN’s operations 101

[key-101]

[Key-101] Lucas eating the profits from his lemonade stand http://twitpic.com/2j05lu
29 minutes ago via Seesmic for iPhone
OSN’s operations 101

[key-146, key-121, key-105, key-101]
Let’s see what’s going on in the world

[key-146, key-121, key-105, key-101]
Let’s see what’s going on in the world

1) FETCH [inbox [from server A]]

[key-146, key-121, key-105, key-101]

*tottycojones* Lucas eating the profits from his lemonade stand [http://twitpic.com/2j05lU](http://twitpic.com/2j05lU)
29 minutes ago via Seesmic for iPhone

*timoreilly* Interested that gmail is smart enough to be showing me ads about electronic health record systems. Wondering what showed them my interest.

about 2 hours ago via Seesmic for iPhone

*pabloryr* I'm at The Taj Mahal. [http://4sq.com/d7YjyM](http://4sq.com/d7YjyM)
about 3 hours ago via foursquare
OSN’s operations 101

Let’s see what’s going on in the world

1) FETCH – inbox [from server]
R= [key-146, key-121, key-105, key-101]

[key-146, key-121, key-105, key-101]
OSN’s operations 101

Let’s see what’s going on in the world

1) FETCH – inbox [from server A]
   R= [key-146, key-121, key-105, key-101]

2) FETCH text WHERE key IN R [from server ]

[key-146, key-121, key-105, key-101]
OSN’s operations 101

Let’s see what’s going on in the world

1) FETCH – inbox [from server A ]
   R = [key-146, key-121, key-105, key-101]

2) FETCH text WHERE key IN R [from server ? ]

Where is the data?

[key-146, key-121, key-105, key-101]
OSN’s Operations 101

1) FETCH – inbox [from server A ]
   R= [key-146, key-121, key-105, key-101]

2) FETCH text WHERE key IN R [from server ? ]

   in A , then data locality!
OSN’s Operations 101

1) FETCH – inbox [from server A]
   R = [key-146, key-121, key-105, key-101]

2) FETCH text WHERE key IN R [from server ?]

in B, F, E, , then no data locality!
### OSN’s operations 101

<table>
<thead>
<tr>
<th>1) Relational Databases</th>
<th>Selects and joins across multiple shards of the database are possible but performance is poor (e.g. MySQL Cluster, Oracle Rack)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="mySql.png" alt="MySQL" /></td>
<td><img src="sqlServer.png" alt="SQL Server" /></td>
</tr>
<tr>
<td><strong>2) Key-Value Stores (DHT)</strong></td>
<td>More efficient than relational databases: multi-get primitives to transparently fetch data from multiple servers.</td>
</tr>
<tr>
<td><img src="mongodb.png" alt="mongoDB" /></td>
<td><img src="redis.png" alt="Redis" /></td>
</tr>
</tbody>
</table>
| ![Cassandra](cassandra.png) | **But it’s not a silver bullet:**  
  - lose SQL query language => programmatic queries  
  - lose abstraction from data operations  
  - suffer from high traffic, eventually affecting performance:  
    - Incast issue  
    - Multi-get hole  
    - latency dominated by the worse performing server |
Maintaining data locality
Maintaining data locality

Users are assigned to servers randomly (DHT-like)
Maintaining data locality

Random Partitioning – DHT
Maintaining data locality

Random Partitioning – DHT
Maintaining data locality

Random Partitioning – DHT
How do we enforce data locality?
Maintaining data locality

Random Partitioning – DHT
How do we enforce data locality?
Replication
Maintaining data locality
Maintaining data locality

#4 Master + Replica
Maintaining data locality

That does it for user #3, but what about the others?
Maintaining data locality

That does it for user #3, but what about the others?

The very same procedure…
Maintaining data locality

After a while…

Data locality for reads for any user is guaranteed!
Maintaining data locality

Data locality for reads for any user is guaranteed!
Maintaining data locality

Data locality for reads for any user is guaranteed!
Maintaining data locality

Data locality for reads for any user is guaranteed!

Random partition + Replication can lead to high replication overheads!

This is full replication!!
Maintaining data locality

Data locality for reads for any user is guaranteed!

Random partition + Replication can lead to high replication overheads!

Can we do better?
Maintaining data locality

We can leverage the underlying social structure for the partition...

Data locality for reads in any user is guaranteed!

Random partition + Replication can lead to high replication overheads!
Maintaining data locality

We can leverage the underlying social structure for the partition...

Data locality for reads in any user is guaranteed!

Random partition + Replication can lead to high replication overheads!
Maintaining data locality

Data locality for reads in any user is guaranteed!

Random partition + Replication can lead to high replication overheads!

We can leverage the underlying social structure for the partition...

Social Partition
Maintaining data locality

We can leverage the underlying social structure for the partition...

Data locality for reads in any user is guaranteed!

Random partition + Replication can lead to high replication overheads!

Social Partition and Replication: SPAR
Maintaining data locality

We can leverage the underlying social structure for the partition...

Data locality for reads in any user is guaranteed!

Random partition + Replication can lead to high replication overheads!

Social Partition and Replication: SPAR

… only two replicas to achieve data locality
Maintaining data locality

We can leverage the underlying social structure for the partition...

Data locality for reads in any user is guaranteed!

Random partition + Replication can lead to high replication overheads!

Social Partition and Replication: SPAR

... only two replicas to achieve data locality
SPAR Algorithm from 10000 feet

“All partition algorithms are equal, but some are more equal than others”
SPAR Algorithm from 10000 feet

“All partition algorithms are equal, but some are more equal than others”

| 1) Online (incremental) | a) Dynamics of the SN (add/remove node, edge)  
b) Dynamics of the System (add/remove server) |
|-------------------------|--------------------------------------------------------------------------------------------------|
| 2) Fast (and simple)    | a) Local information  
b) Hill-climbing heuristic  
c) Load-balancing via back-pressure |
| 3) Stable               | a) Avoid cascades |
| 4) Effective            | a) Optimize for MIN_REPLICA (*i.e.* NP-Hard)  
b) While maintaining a fixed number of replicas per user for redundancy (e.g. K=2) |
Looks good on paper, but...

... let’s try it for real

Real OSN data

• Twitter
  – 2.4M users, 48M edges, 12M tweets for 15 days (Twitter as of Dec08)
• Orkut (MPI)
  – 3M users, 223M edges
• Facebook (MPI)
  – 60K users, 500K edges

Partition algorithms

• Random (DHT)
• METIS
• MO+
  • modularity optimization + hack
• SPAR online
## How many replicas are generated?

<table>
<thead>
<tr>
<th></th>
<th>Twitter 16 partitions</th>
<th>Twitter 128 partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rep. overhead</td>
<td>% over K=2</td>
</tr>
<tr>
<td><strong>SPAR</strong></td>
<td>2.44</td>
<td>+22%</td>
</tr>
<tr>
<td><strong>MO+</strong></td>
<td>3.04</td>
<td>+52%</td>
</tr>
<tr>
<td><strong>METIS</strong></td>
<td>3.44</td>
<td>+72%</td>
</tr>
<tr>
<td><strong>Random</strong></td>
<td>5.68</td>
<td>+184%</td>
</tr>
</tbody>
</table>
How many replicas are generated?

<table>
<thead>
<tr>
<th></th>
<th>Twitter 16 partitions</th>
<th>Twitter 128 partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rep. overhead</td>
<td>% over K=2</td>
</tr>
<tr>
<td>SPAR</td>
<td>2.44</td>
<td>+22%</td>
</tr>
<tr>
<td>MO+</td>
<td>3.04</td>
<td>+52%</td>
</tr>
<tr>
<td>METIS</td>
<td>3.44</td>
<td>+72%</td>
</tr>
<tr>
<td>Random</td>
<td>5.68</td>
<td>+184%</td>
</tr>
</tbody>
</table>

2.44 replicas per user (on average)
2 to guarantee redundancy
+ 0.44 to guarantee data locality (+22%)
How many replicas are generated?

<table>
<thead>
<tr>
<th></th>
<th>Twitter 16 partitions</th>
<th></th>
<th>Twitter 128 partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rep. overhead</td>
<td>% over K=2</td>
<td>Rep. overhead</td>
</tr>
<tr>
<td>SPAR</td>
<td>2.44</td>
<td>+22%</td>
<td>3.69</td>
</tr>
<tr>
<td>MO+</td>
<td>3.04</td>
<td>+52%</td>
<td>5.14</td>
</tr>
<tr>
<td>METIS</td>
<td>3.44</td>
<td>+72%</td>
<td>8.03</td>
</tr>
<tr>
<td>Random</td>
<td>5.68</td>
<td>+184%</td>
<td>10.75</td>
</tr>
</tbody>
</table>

Replication with random partitioning, too costly!
How many replicas are generated?

<table>
<thead>
<tr>
<th></th>
<th>Twitter 16 partitions</th>
<th>Twitter 128 partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rep. overhead</td>
<td>% over K=2</td>
</tr>
<tr>
<td>SPAR</td>
<td>2.44</td>
<td>+22%</td>
</tr>
<tr>
<td>MO+</td>
<td>3.04</td>
<td>+52%</td>
</tr>
<tr>
<td>METIS</td>
<td>3.44</td>
<td>+72%</td>
</tr>
<tr>
<td>Random</td>
<td>5.68</td>
<td>+184%</td>
</tr>
</tbody>
</table>

Other social based partitioning algorithms have higher replication overhead than SPAR.
The SPAR architecture
The SPAR architecture

Application, 3-tiers
The SPAR architecture

Application, 3-tiers

SPAR Middleware (data-store specific)
The SPAR architecture

SPAR Controller: Partition manager and Directory Service

SPAR Middleware: (data-store specific)

Application, 3-tiers
SPAR in the wild

• **Twitter clone** (Laconica, now Statusnet)
  – Centralized architecture, PHP + MySQL/Postgres

• **Twitter data**
  – Twitter as of end of 2008, 2.4M users, 12M tweets in 15 days

• **The little engines**
  – 16 commodity desktops: Pentium Duo at 2.33Ghz, 2GB RAM connected with Gigabit-Ethernet switch

• **Test SPAR on top of:**
  – MySQL (v5.5)
  – Cassandra (v0.5.0)
SPAR on top of MySQL

- Different read request rate (application level):
  - 1 call to LDS, 1 join to on the notice and notice inbox to get the last 20 tweets and its content
  - User is queried only once per session (4min)

<table>
<thead>
<tr>
<th></th>
<th>MySQL Full-Replication</th>
<th>SPAR + MySQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>99th &lt; 150ms</td>
<td>16 req/s</td>
<td>2500 req/s</td>
</tr>
</tbody>
</table>

- Spar allows for data to be partitioned, improving both query times and cache hit ratios.
- Full replication suffers from low cache hit ratio and large tables (1B+)
SPAR on top of Cassandra

• Different read request rate (application level):

<table>
<thead>
<tr>
<th></th>
<th>Vanilla Cassandra</th>
<th>SPAR + Cassandra</th>
</tr>
</thead>
<tbody>
<tr>
<td>99th &lt; 100ms</td>
<td>200 req/s</td>
<td>800 req/s</td>
</tr>
</tbody>
</table>

Network bandwidth is not an issue but Network I/O and CPU I/O are:

– Network delays is reduced
  • Worse performing server produces delays

– Memory hit ratio is increased
  • Random partition destroys correlations
Problem
Solution
Implementation
Conclusions
Conclusions

SPAR provides the means to achieve transparent scalability

1) For applications using RDBMS (not necessarily limited to OSN)

2) And, a performance boost for key-value stores due to the reduction of Network I/O
Thanks!

jmps@tid.es
http://research.tid.es/jmps/
@solso