SPAR

The Little Engine(s) That Could: Scaling Online Social Networks

Arman Idani

28 Feb 2012

R202 – Data Centric Networking


facebook
December 2010

Background

- Social Networks are hugely interconnected
- Scaling interconnected networks is difficult
  - Data locality
  - Network traffic
  - Programming semantics
- Social networks grow significantly in a short period of time
  - Twitter grew ~15x in a month (Early 2009)

How to Scale OSNs?

- Horizontal scaling
  - Cheap commodity servers
  - Amazon EC2, Google AppEngine, Windows Azure
- How to partition the data?
  - The actual data and replicas
- Application scalability?

Designer’s Dilemma

- Commit resources to adding features to OSNs?
  - Appealing features and attracts new users
  - Might not scale in the same pace as users’ demand
  - Death-by-success scenario (e.g. Friendster)

- Make a scalable system first and then add features
  - High developer resource
  - Might not compete well if competitors are richer feature-wise
  - No death-by-success

Data Partitioning

- Random partitioning and replication (DHT)
  - Locality of interconnected data not preserved
  - High network workload
  - Deployed by Facebook and Twitter

- Full replication
  - Lower network workload
  - High server/user requirement

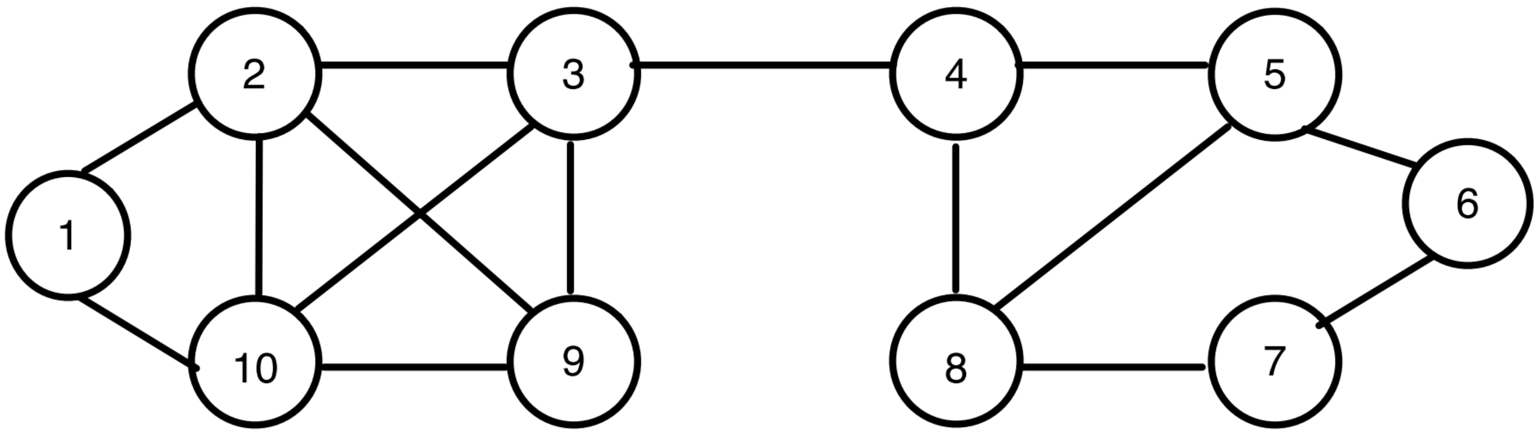
Solution?

- How to achieve application scalability?
  - Preserve locality for all of the data relevant to the user
  - Local programming semantics for applications

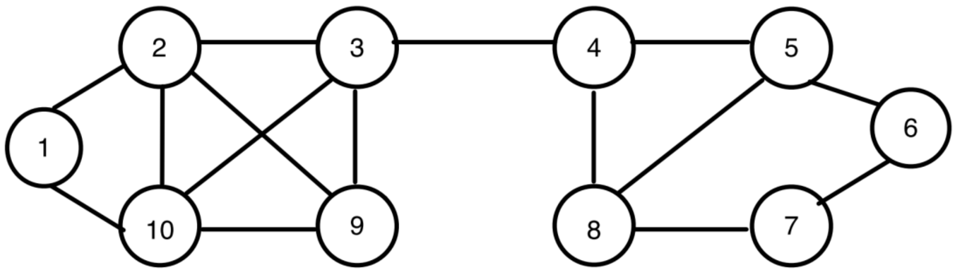
SPAR

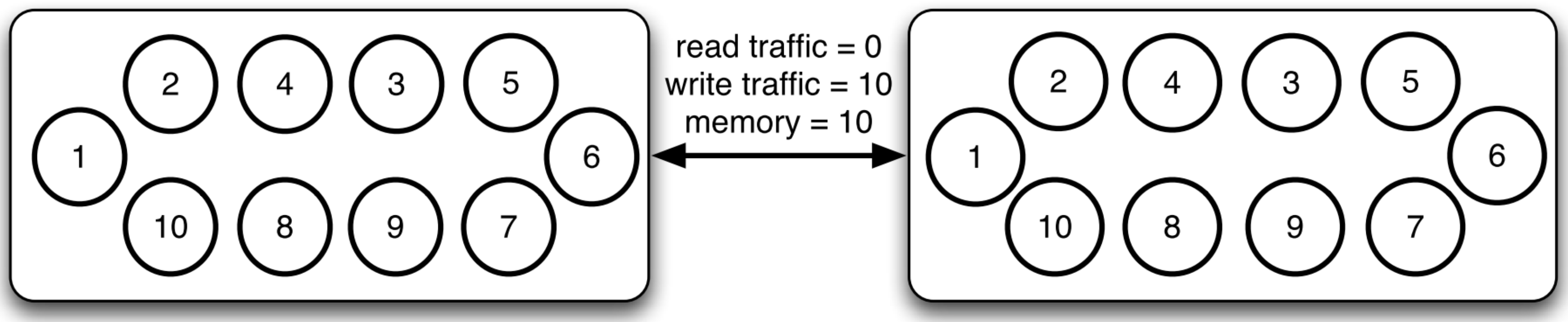
- Replicas of all friend data on the same server
  - Local queries to the data
  - Illusion that OCN is running on a centralized server
- No network bottleneck
- Support for both relational databases and key-value stores

Example (ONS)

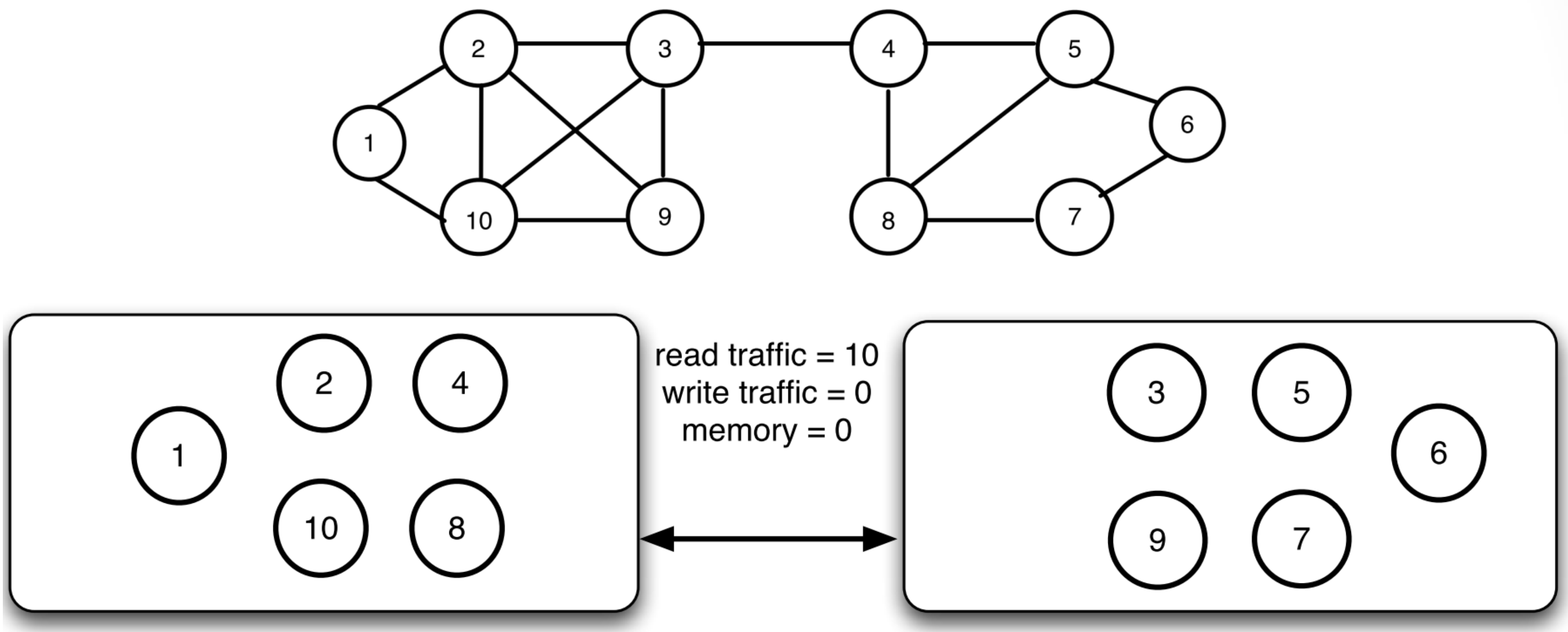


Full Replication

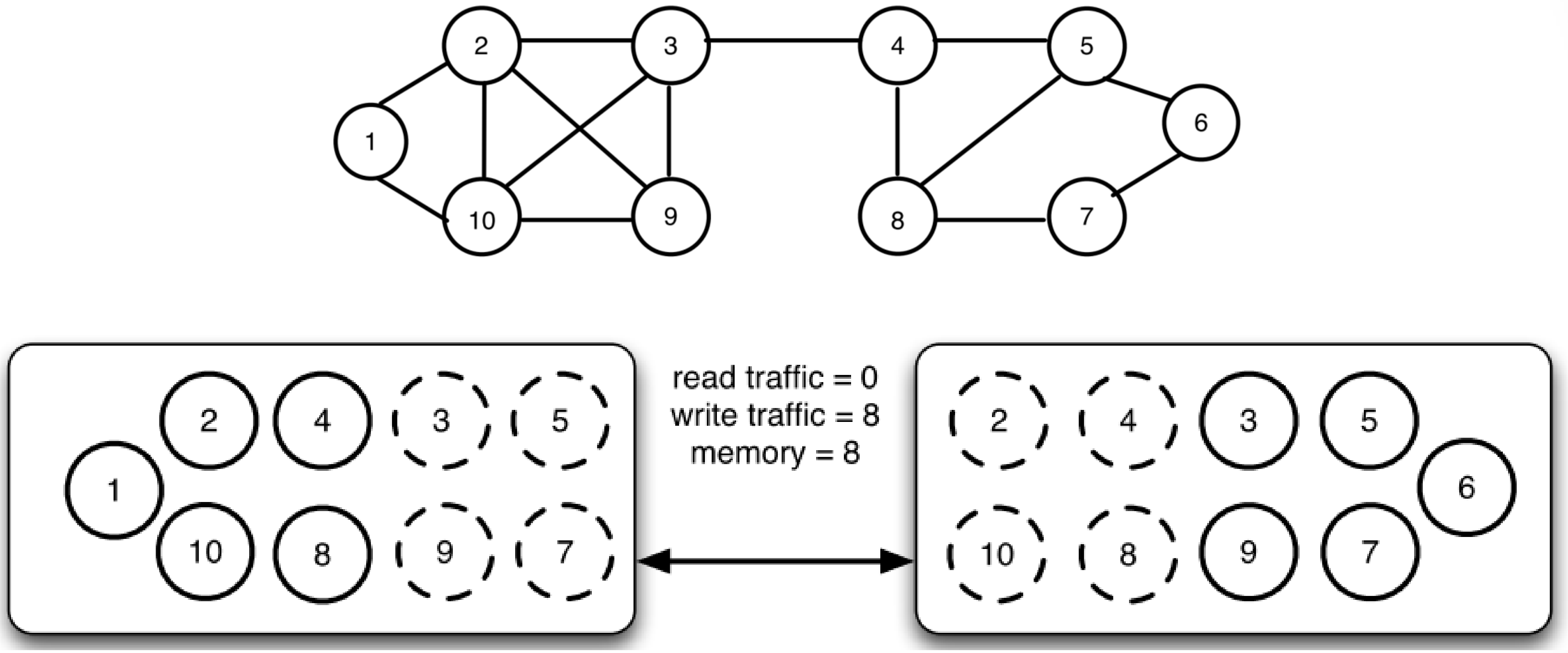




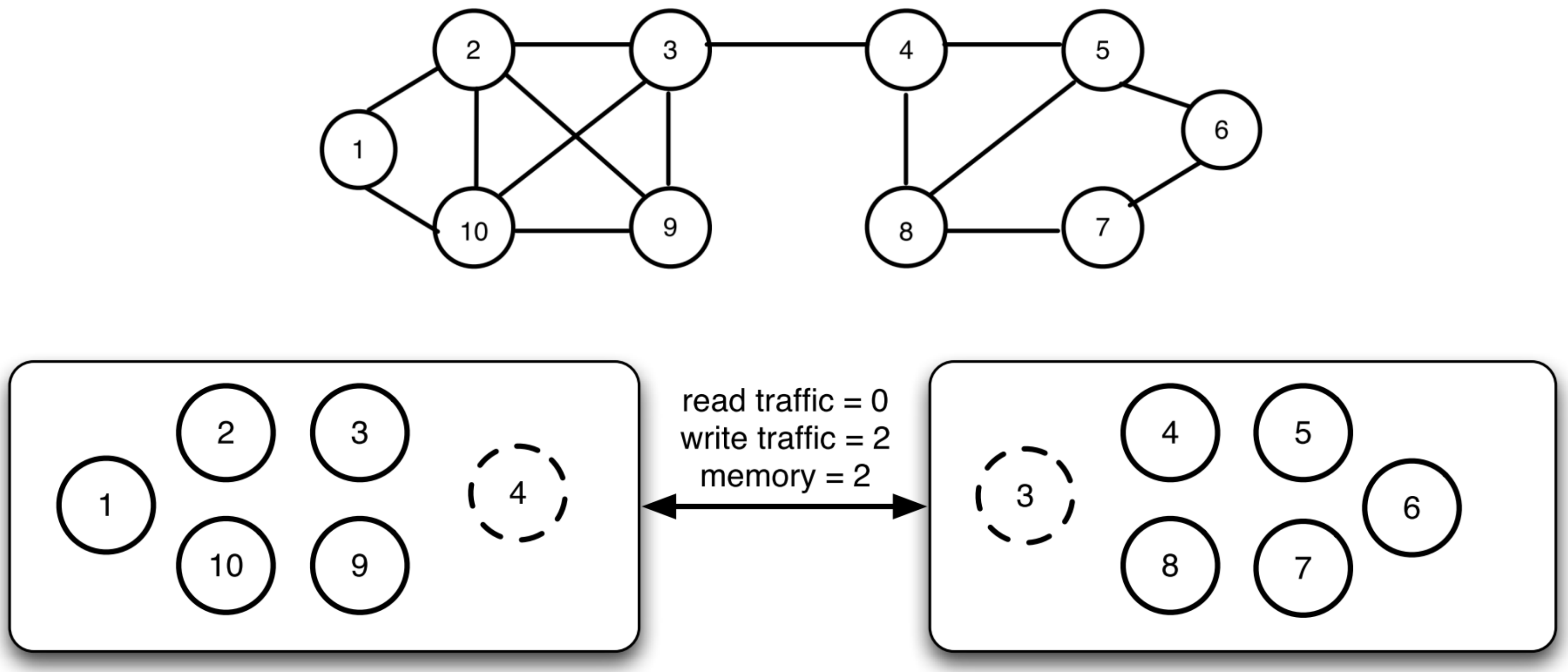
DHT

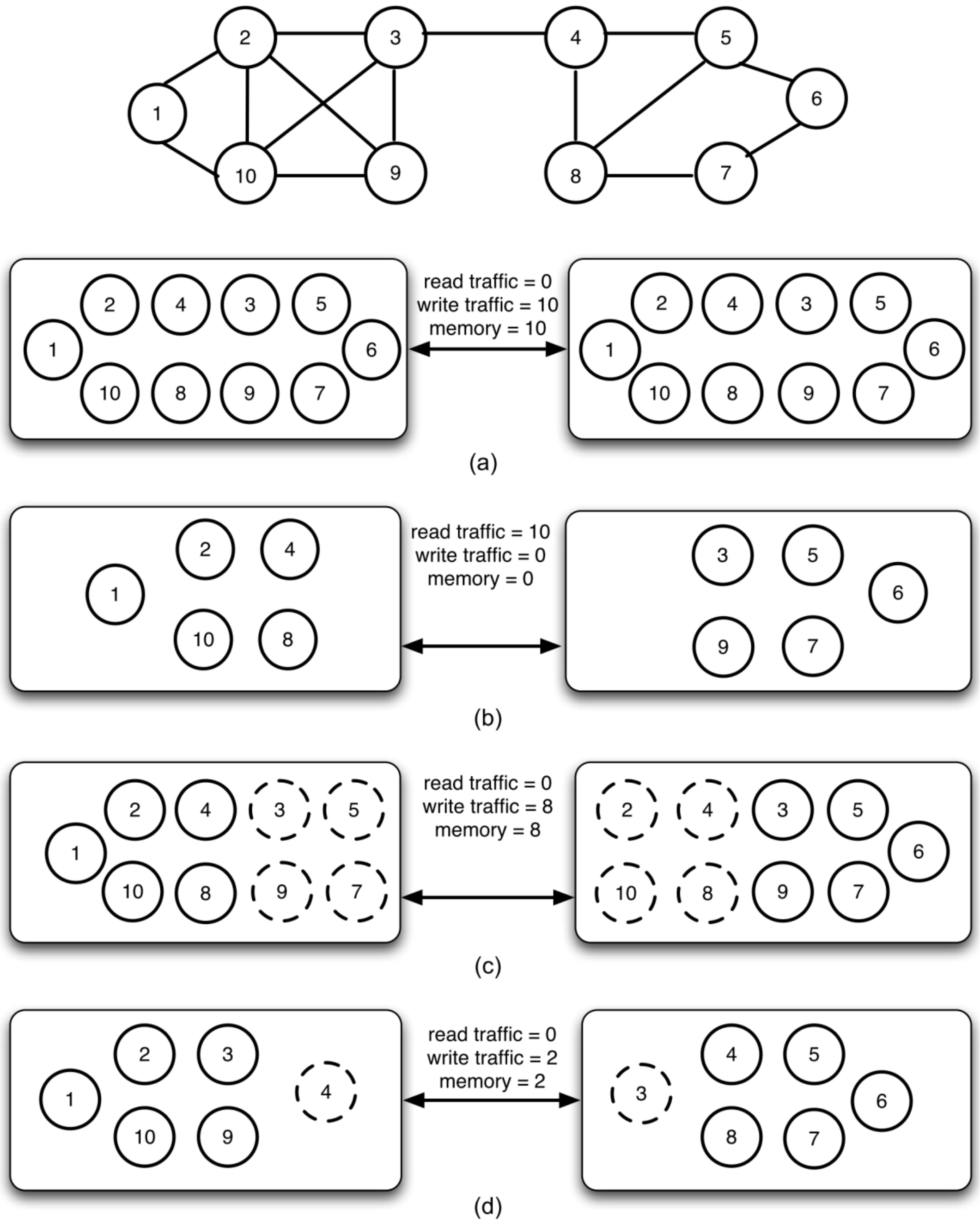


DHT + Neighbour Replication



SPAR




read traffic = 0
write traffic = 10
memory = 10
(a)
read traffic = 10
write traffic = 0
memory = 0
(b)
read traffic = 0
write traffic = 8
memory = 8
(c)
read traffic = 0
write traffic = 2
memory = 2
(d)

SPAR Requirements

- Maintain local semantics
- Balance loads
- Machine failure robustness
- Dynamic online operations
- Be stable
- Minimize replication overhead

Partition Management

- Partition Management in six events:
  - Node/Edge/Server
  - Addition/Removal

- Edge addition
  - Configuration 1: exchange slave replicas
  - Configuration 2: move the master

- Server addition
  - Option 1: Redistribute the masters to the new server
  - Option 2: Let it fill by itself

Implementation

- SPAR is a middle-ware between datacenter and application
- Applications developed as if centralized

- Four SPAR components:
  - Directory Service
  - Local Directory Service
  - Partition Manager
  - Replication Manager

DS and LDS

- Directory Service
  - Handles data distribution
  - Knows about location of master and slave replicas
  - Key-table lookup
- Local Directory Service
  - Only access to a fraction of key-table
  - Acts as a cache

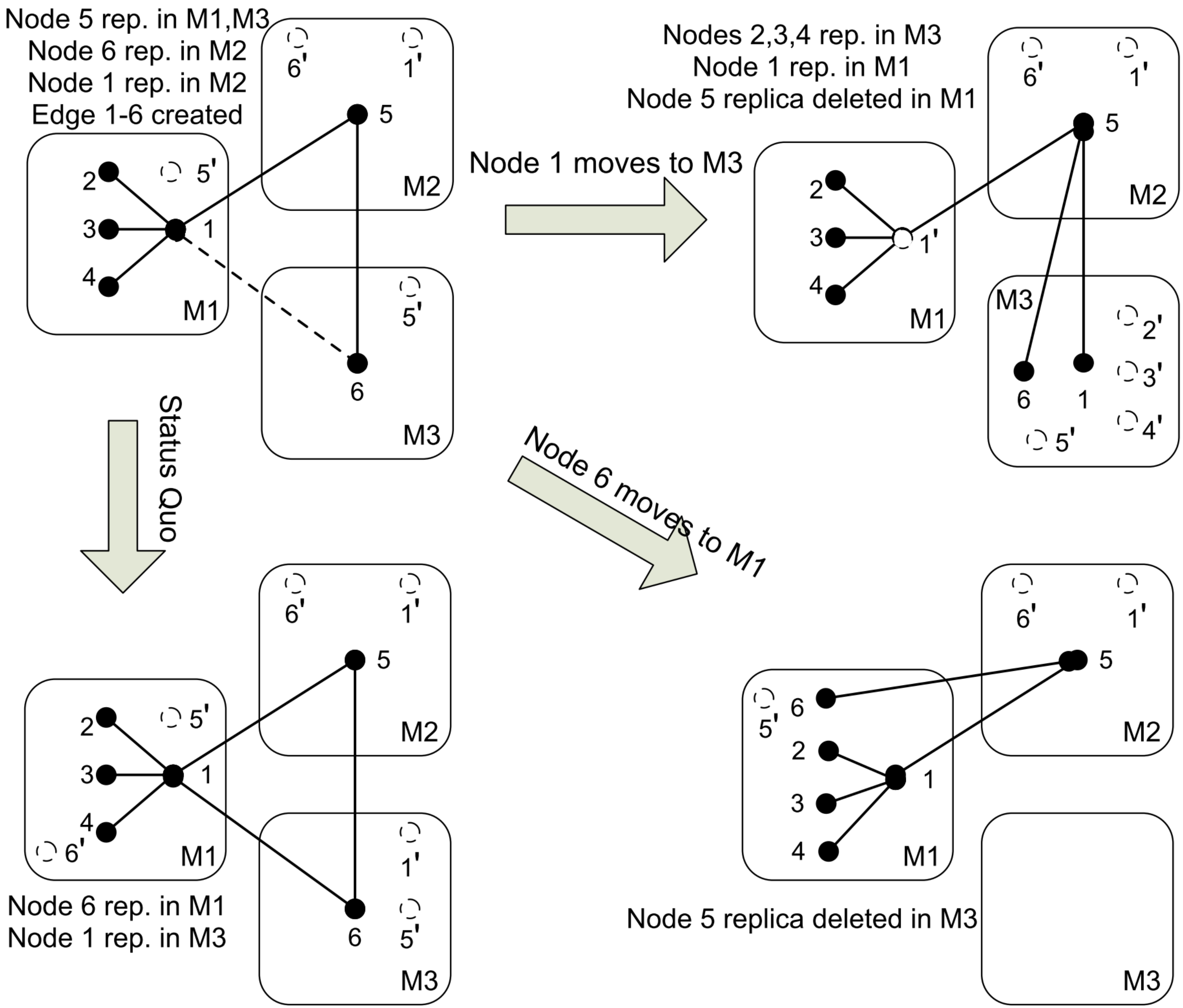
Partition Manager

- Maps the users' keys to replicas
- Schedules movement of replicas
- Redistributes replicas in case of server addition/removal
- Can be both centralized or distributed
- Reconciliation after data movements
  - Version-based (Similar to Amazon Dynamo)
- Handling failures
  - Permanent or transient

Replication Manager

- Propagates updates to replicas
  - Updates are queries
  - Propagates queries, not data

EXAMPLE!


Node 5 rep. in M1,M3
Node 6 rep. in M2
Node 1 rep. in M2
Edge 1-6 created
Node 1 moves to M3
Nodes 2,3,4 rep. in M3
Node 1 rep. in M1
Node 5 replica deleted in M1
Status Quo
Node 6 moves to M1
Node 6 rep. in M1
Node 1 rep. in M3
Node 5 replica deleted in M3
M1
M2
M3

Evaluation

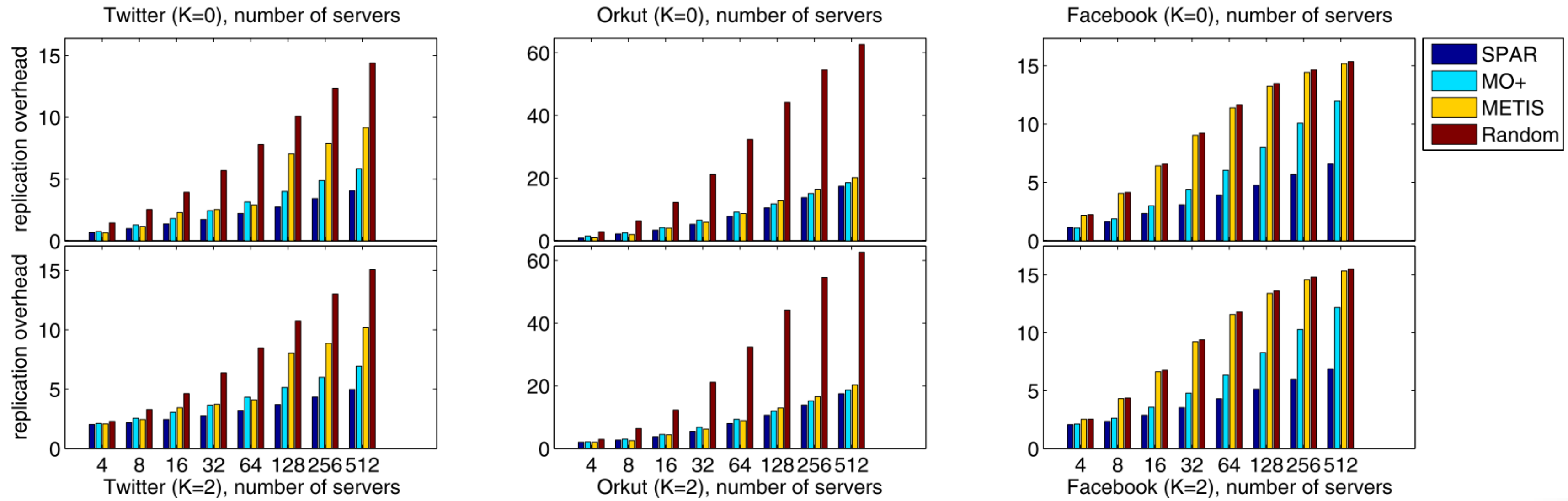
- Measurement driven evaluation
  - Replication overhead
  - K-redundancy requirement

- Twitter
  - 12m tweets by 2.4m users (50% of twitter)
- Facebook
  - 60k users, 1.5m friendships
- Orkut
- 3m users, 224m friendships

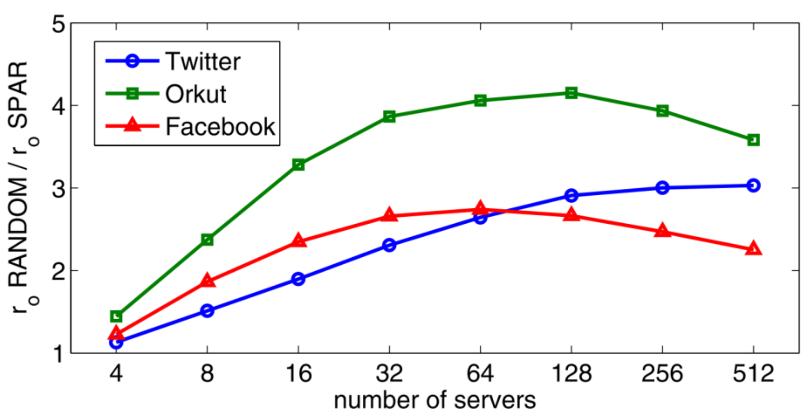
Vs.

- Random Partitioning
  - Solutions deployed by Facebook, Twitter
- METIS
  - Graph Partitioning (offline)
  - Focus on minimizing inter-partition edges
- Modularity Optimizations (MO+)
  - Community detection

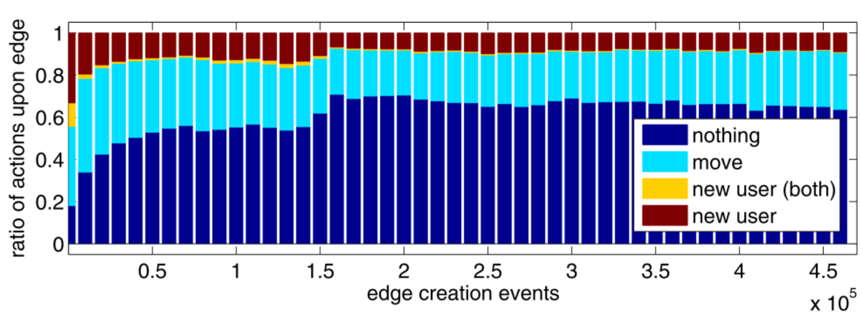
Results



Twitter (K=0), number of servers
Orkut (K=0), number of servers
Facebook (K=0), number of servers
replication overhead
replication overhead
SPAR
MO+
METIS
Random
4 8 16 32 64 128 256 512
Twitter (K=2), number of servers
Orkut (K=2), number of servers
Facebook (K=2), number of servers



Twitter
Orkut
Facebook
r_o RANDOM / r_o SPAR
4 8 16 32 64 128 256 512
number of servers



ratio of actions upon edge
nothing
move
new user (both)
new user
edge creation events
x 10^5

Twitter Analysis

- Twitter (12m tweets by 2.4m users), K=2, M=128
  - Average replication overhead: 3.6
  - 75% have 3 replicas
  - 90% < 7
  - 99% < 31
  - 139 users (0.006%) on all servers

Adding Servers

- Option 1: wait for arrivals to fill in
  - 16 to 32 Servers
  - Replication overhead: 2.78
  - 2.74 if started with 32
- Option 2: redistribution all nodes
  - Overhead: 2.82

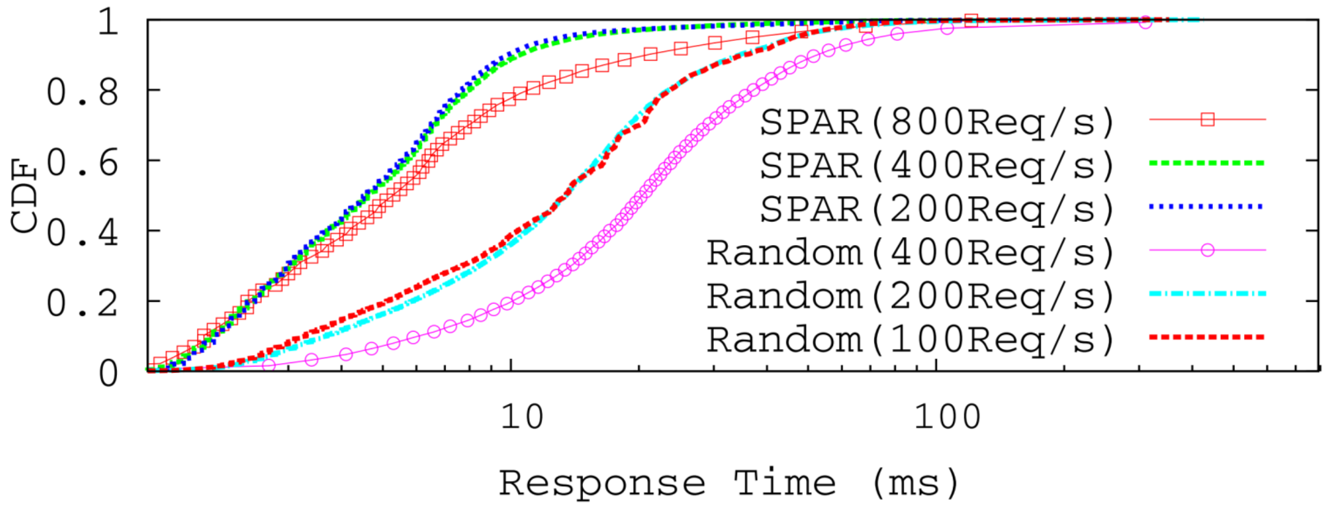
Removing Servers

- Removal of one server
  - 500k (20%) movement of nodes
  - A very high penalty, but not common to scale down the network
- Transient removal of servers (fault)
  - Temporarily assign a slave replica as master
  - No locality requirement
  - Wait for the failed server to come back and restore

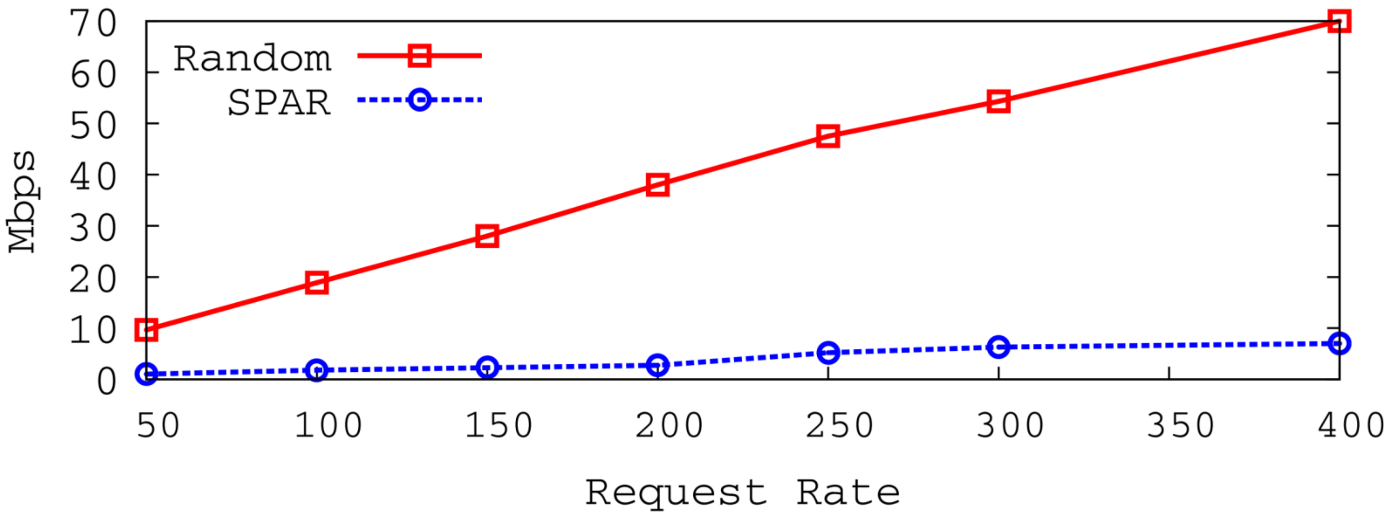
SPAR in the Wild

- Apache Cassandra (key-value)
  - Random Partitioning
- MySQL (relational database)
  - Full replication
  - Not feasible to even try
- 16 commodity servers
  - Pentium Duo 2.33
  - 2GB RAM
  - Single HDD

Response Times



Network Activity



SPAR (+)

- Scales well and easily
- Local programming semantics
- Low network traffic (when running apps)
- Low latency
- Fault tolerance
- No designer’s dilemma

SPAR (-)

- Assumption: All relevant data are one-hop away
  - Is it true? Maybe not
  - To maintain locality of two hops, replication overhead will be increased exponentially
- No support for privacy
  - Users have different privacy settings for different users, so replicas of each user for each friendship will be different
- Practically no scale-down