How to Configure 3000 High Level Trigger Nodes?

Amedeo Perazzo, Su Dong, Rainer Bartoldus, Sarah Demers, Steffen Luitz

SLAC ATLAS Forum Meeting

Sep 7th 2006, SLAC

Problem

- ~3000 node synchronously download ~100MB data
  - Assuming optimal database performance, db link 10 Gb, all nodes connect to one server, configure transition would take:

$$T \cong 3000 * 100MB / 600MB/s = 8 \text{ min}$$

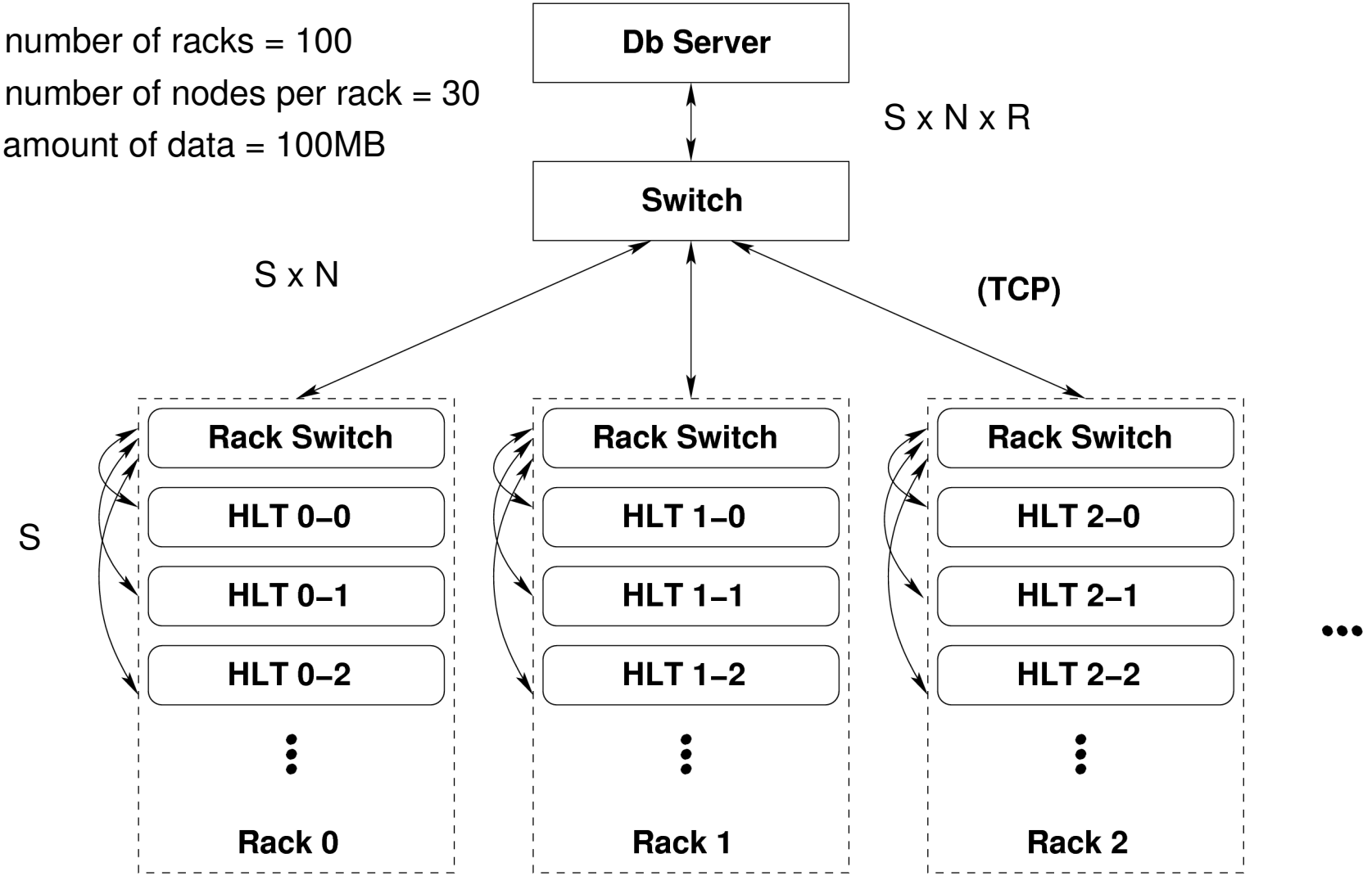
  - Worse by factor 10 if db link is 1Gb
  - Worse by factor C=8 (number of cores per node) if use separate HLT process for each core
- Even best case scenario above pain threshold

HLT Configuration Network

R = number of racks = 100

N = number of nodes per rack = 30

S = amount of data = 100MB



Possible solutions: db replication

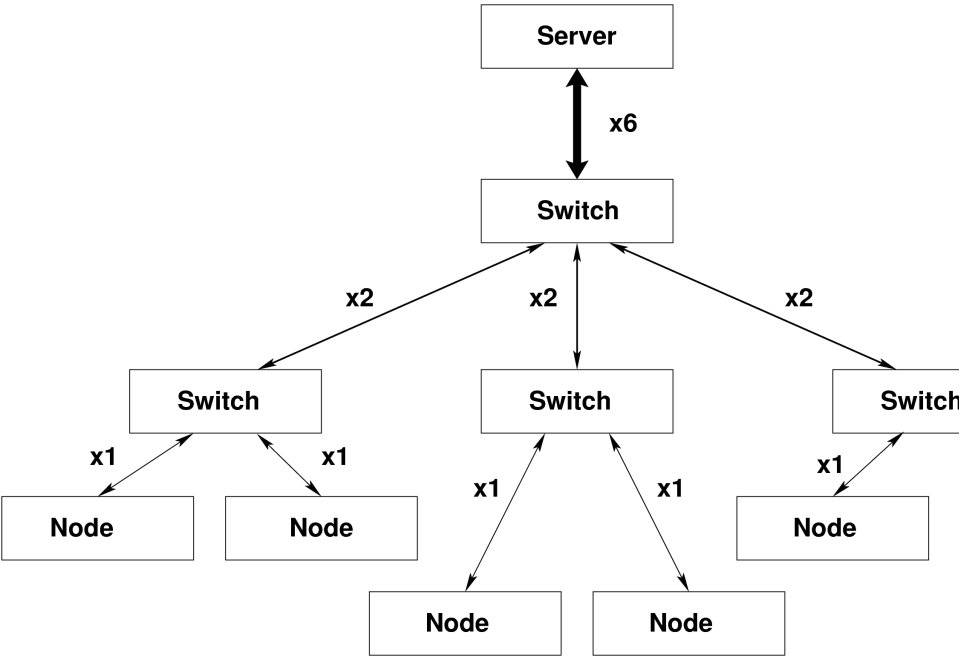
- Database server replication
  - Run multiple MySQL (or Oracle, or SQLite) slave servers at various critical points in the system (eg one server for every rack)
- Updates are pushed to the slave servers when master db contents change
  - No extra custom code, good performance, maintenance nightmare
  - Integration with ATLAS software non optimal

Possible solutions: caching proxy

- Caching proxies
  - ✓ Proxy compares client requests with entries in memory cache, if entry is found it is sent directly to client, otherwise request is forwarded to db server
- Which kind of proxy?
  - ✓ Do available tools provide good enough performance?
  - ✓ Would custom proxies provide better performance?
    - ✗ Positive side effect of custom proxy would be optimal integration of control and error paths with ATLAS TDAQ

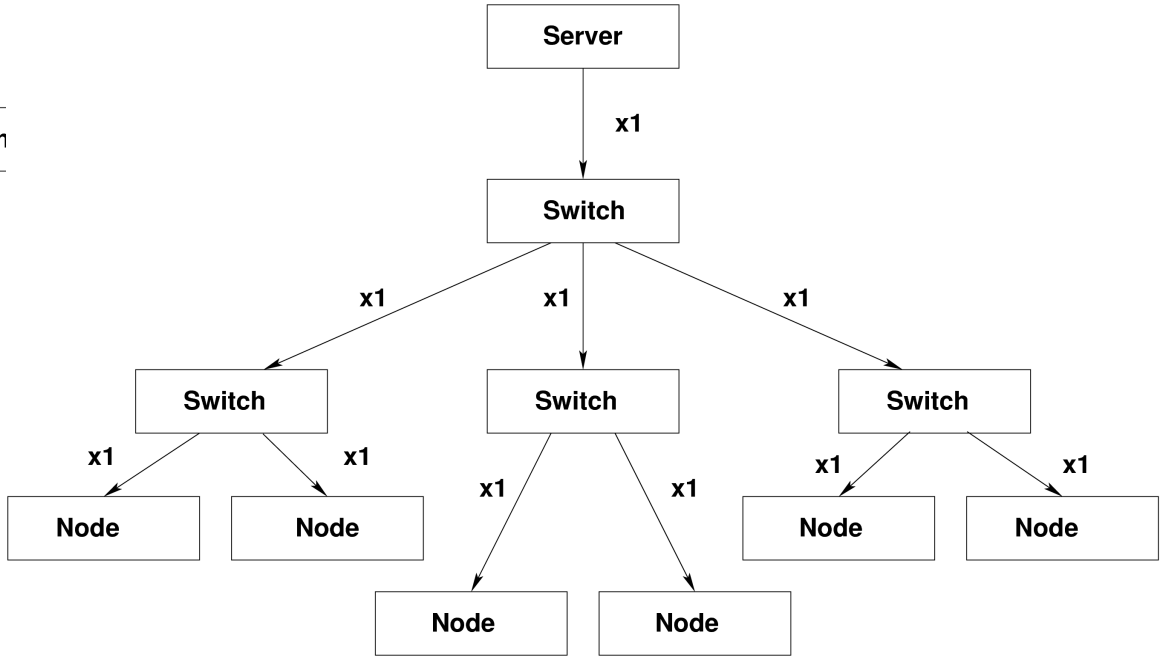
Possible solutions: multicast

- UDP Multicast: High performance, hard to implement reliably, does require custom proxy

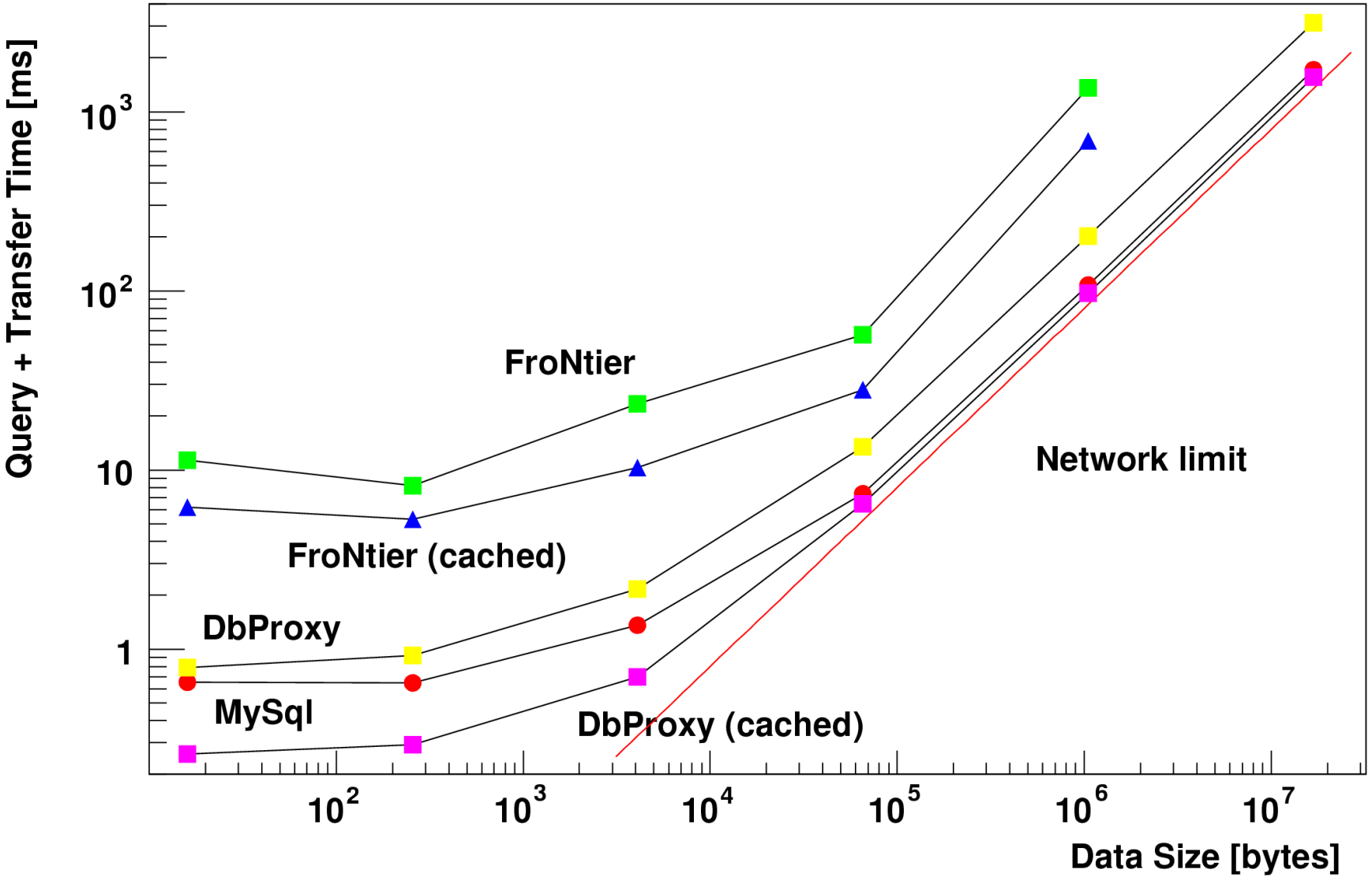


⇐TCP Connections

UDP Multicast (connectionless) ⇒



Performance Studies



Our Proposal

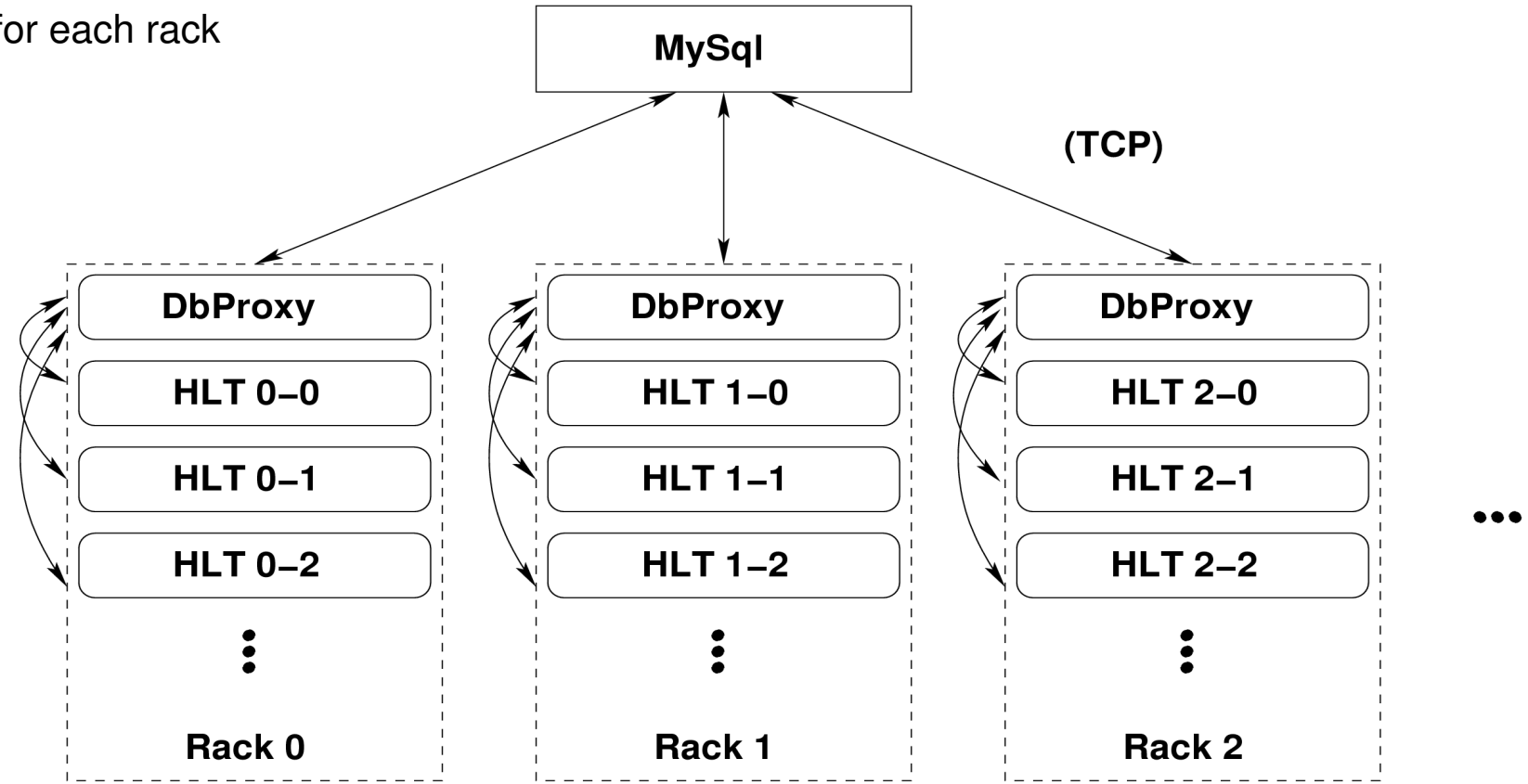
- Custom made caching proxy
  - Language used same as ATLAS TDAQ code: C++
- Using 1 proxy per rack, 1Gb links:

$$T \cong (\max(R, N)+1) * 100MB / 60MB/s \cong 3 \text{ min}$$

  - Faster, but still close to pain threshold
  - This configuration reduces strain on database server by factor N=30

Configuration Example (I)

One layer of proxies: one proxy for each rack



All HLT nodes in one rack communicate with one, per rack, proxy

Our Proposal (II)

- Next step: allow for hierarchical structure of proxies
- Using additional layer of proxies between db and racks, one proxy per rack, 1Gb links:

$$T \cong (\max(L, R', N)+2) * 100MB / 60MB/s$$

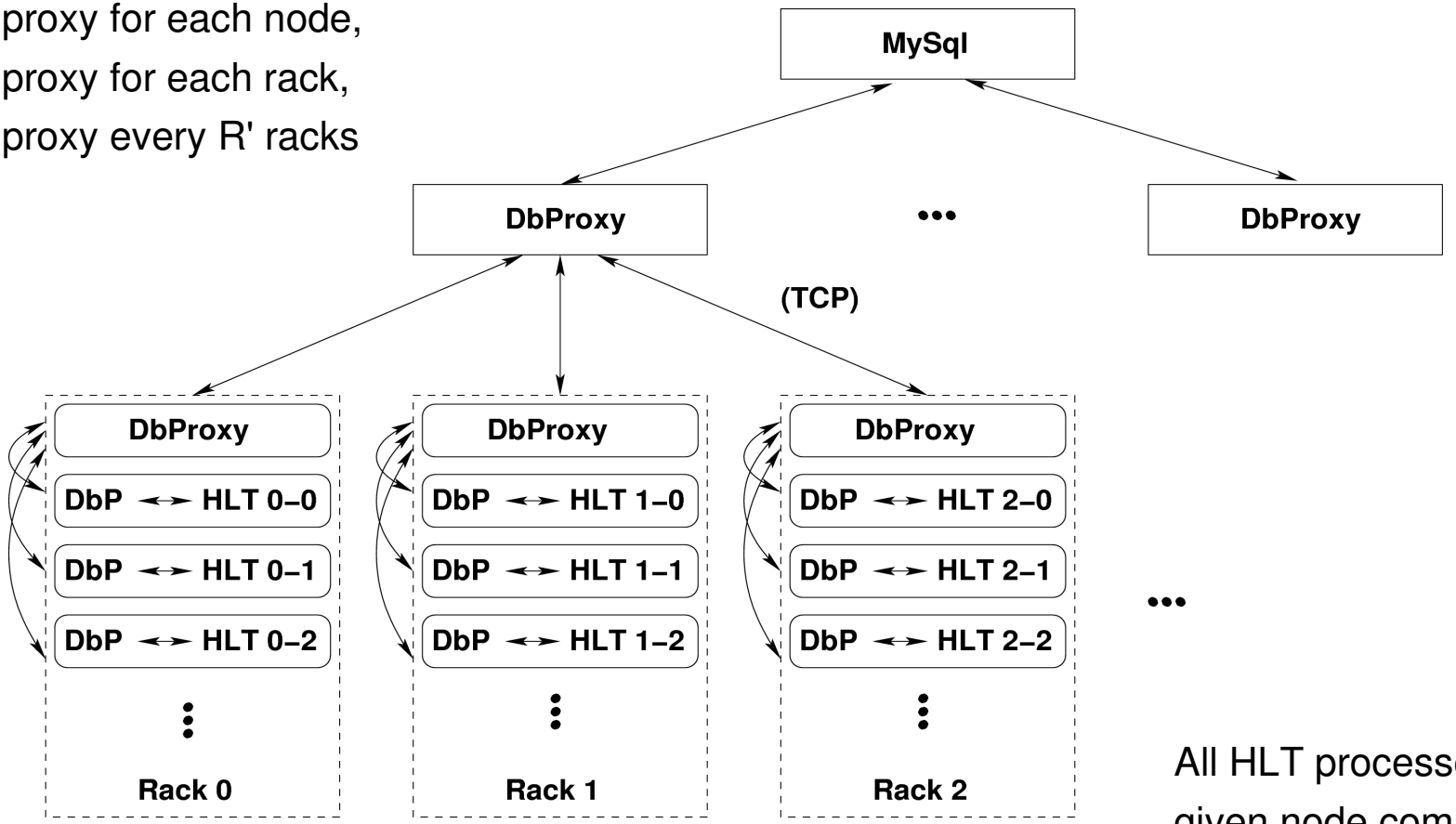
with L=3 (number of level 0 proxies), R'=R/L=30:

$$T \cong 1 \text{ min}$$

  - ➔ Could even afford 1 HLT process per core by introducing 1 proxy per node and still $T \cong 1$ min

Configuration Example (II)

Three layers of proxies:
one proxy for each node,
one proxy for each rack,
one proxy every R' racks



All HLT processes in a given node communicate with one, per node, proxy

Our Proposal (III)

- Next step: ability to multicast data from top level master proxy to lower level proxies
- Very fast, no data replication, maximum parallelism

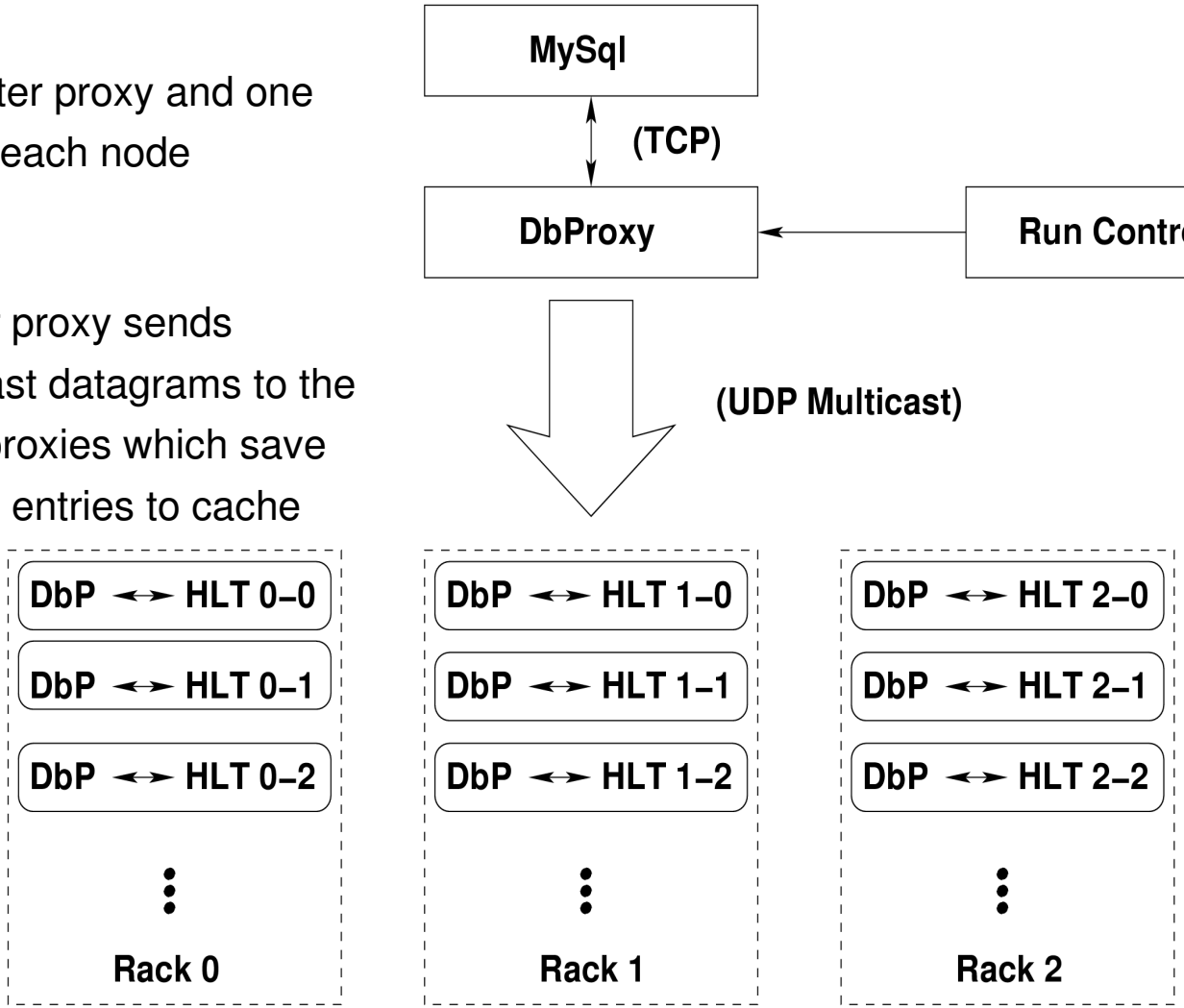
T ~ 100MB / 30MB/s ~ 3 sec

- Multicast doesn't use TCP, but UDP which is not reliable: dropping one ethernet packet can cause entire datagram to be lost
  - First approximation no need to make UDP reliable (proxy would fall back to make TCP requests when entry missing)

Configuration Example (III)

One master proxy and one proxy for each node

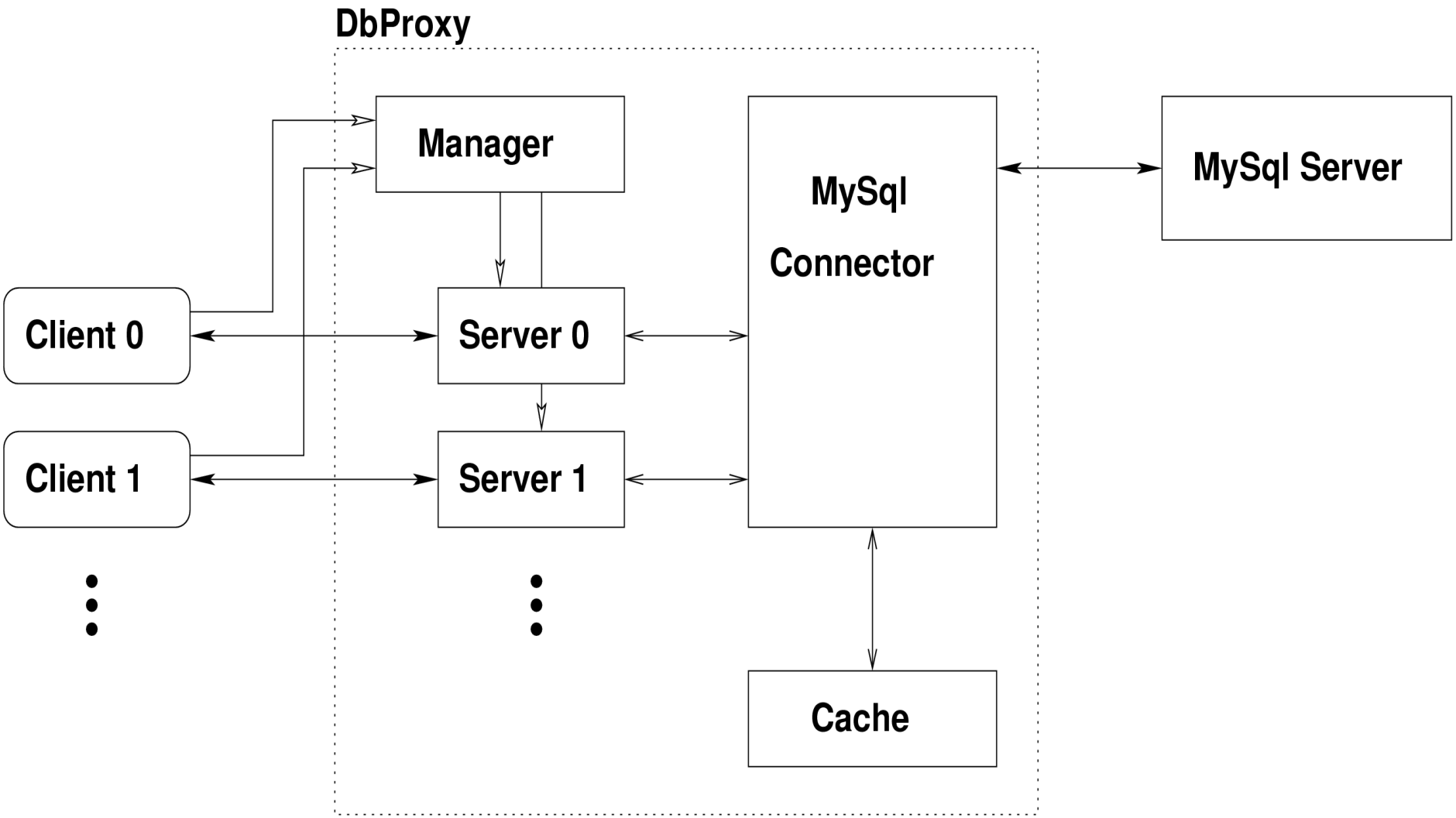
Master proxy sends multicast datagrams to the other proxies which save the DB entries to cache



All HLT processes in a given node communicate with one, per node, proxy

...

Proxy Data Flow



Proxy Implementation

- Separate threads for data and control paths
- Use `select()` to provide serialization for each path and for communication between paths
- Scatter/gather IO to avoid data copy
- No external API (eg MySQL or Oracle) to provide our own memory managment and avoid data copy
- Optimized hash map to implement cache

Conclusions

- Database caching proxy has been built and integrated with ATLAS TDAQ
- Most interesting proxy features are:
  - Fast, small, minimal dependencies on external code
  - Hierarchical configuration
  - Ability to use multicast (still missing reliable UDP though)
- Farm with ~64 nodes being built at SLAC for testing
- Will be used for large scale tests this fall