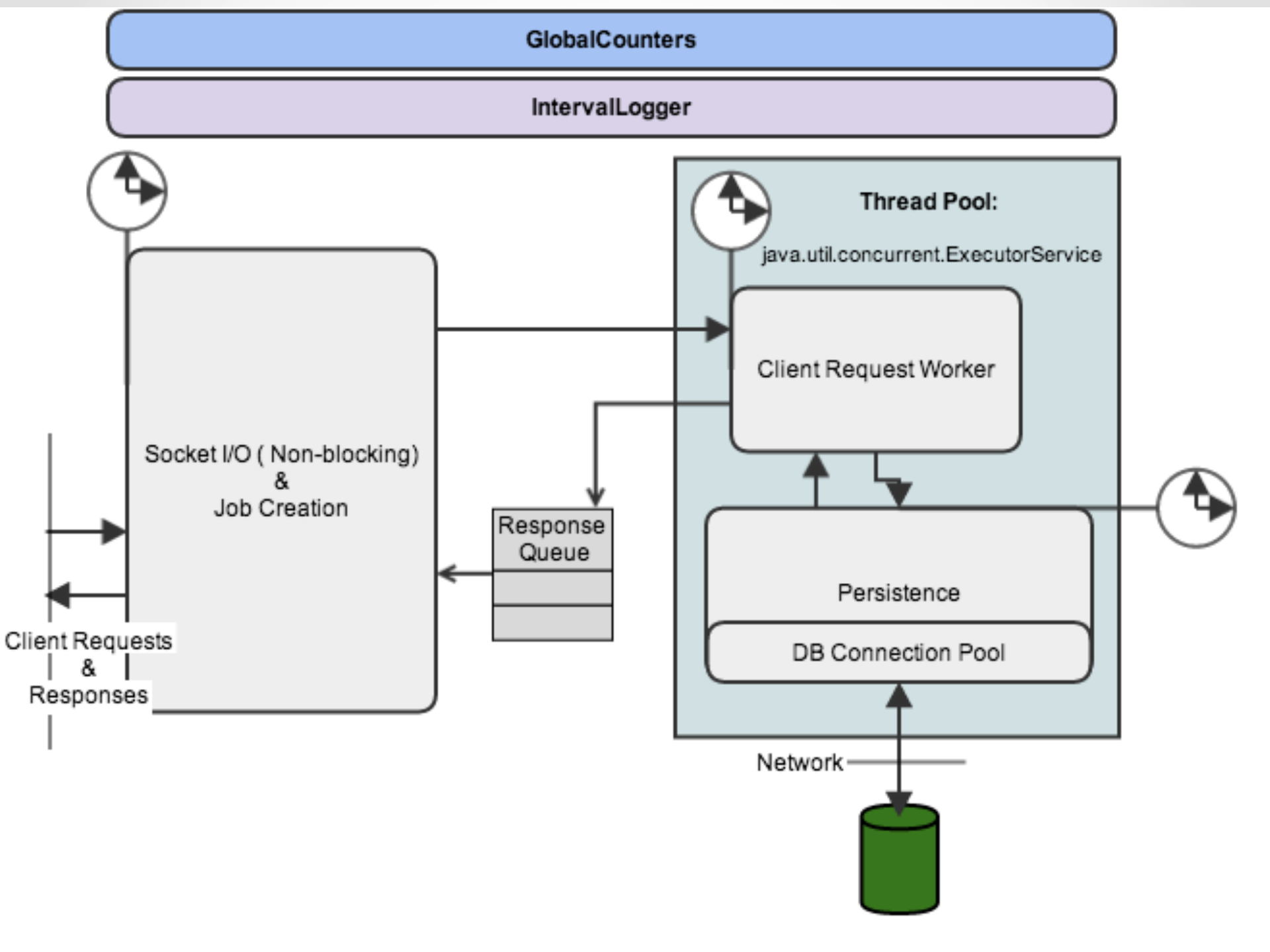
ThorBang MQ

erik jonsson THORén michael BANG

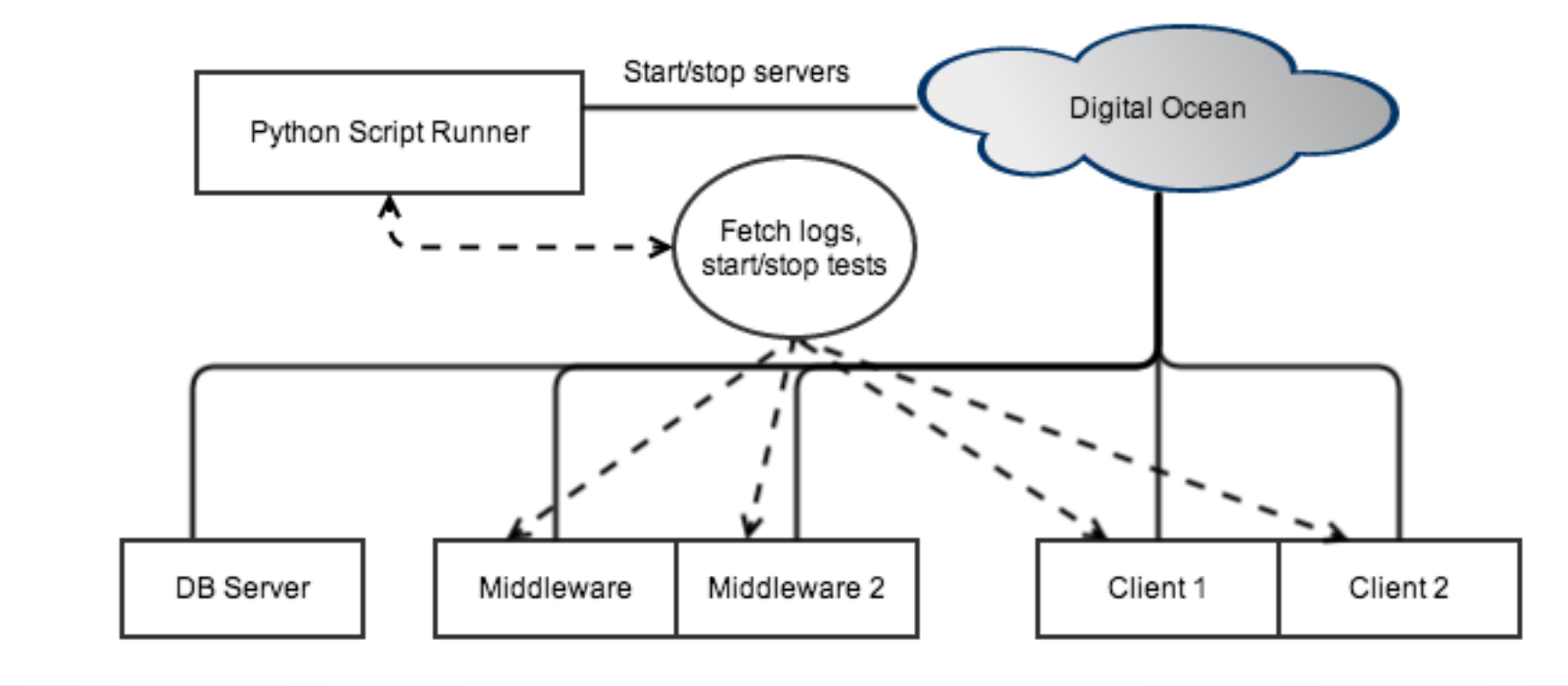
ASL Milestone 1

Agenda

- Design of System
- Design of Testing Setup
- Summary of Experiments
- 4h trace
- Things Learned


GlobalCounters
IntervalLogger
Thread Pool:
java.util.concurrent.ExecutorService
Client Request Worker
Socket I/O (Non-blocking)
&
Job Creation
Response
Queue
Persistence
DB Connection Pool
Client Requests
&
Responses
Network

Design of Testing Setup



Design of System

Produced Log Files

1. Time since server start
2. Push Requests
3. Pop / Peek Requests
4. Total Requests
5. Total Think Time in Socket I/O and job creation
6. Total Think Time in Request Worker
7. Total Think Time in Persistence-component

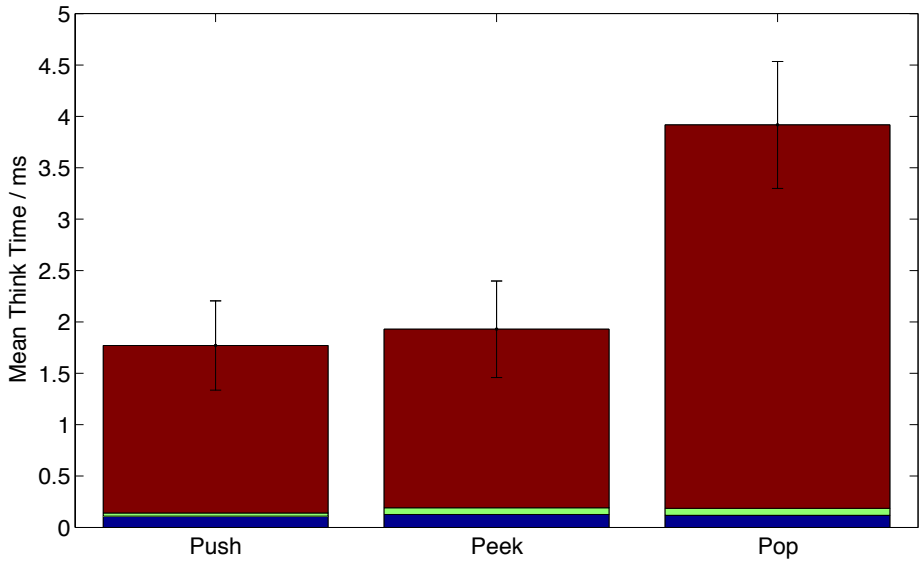
```
593135,1101,1090,2191,47713665384,29913823472,29510444900
594136,1200,1214,2414,47829943204,30080084320,29805427278
```

Experiments conducted

- **Micro-benchmarks**
- **Total number of clients**
- **Number of middleware instances**
- **Number of database connections**
- **Number of worker threads**
- **Frequency of requests**
- Size of messages
- Impact of using database vs. no storage at all
- Size of dataset
- Client threads per client machine
- Number of queues used

Results Of Experiments Conducted

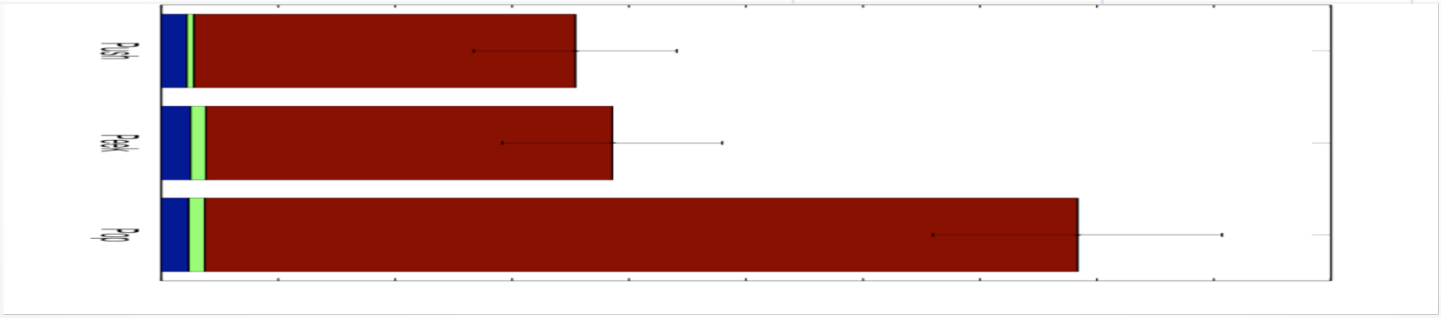
Type of Request	Response Time [ms]
Push Message	1.86 ± [-0.57 , 0.61]
Peek Queue	2.02 ± [-0.67 , 0.70]
Pop Queue	3.99 ± [-0.79 , 1.17]



50 db-connections
50 worker threads

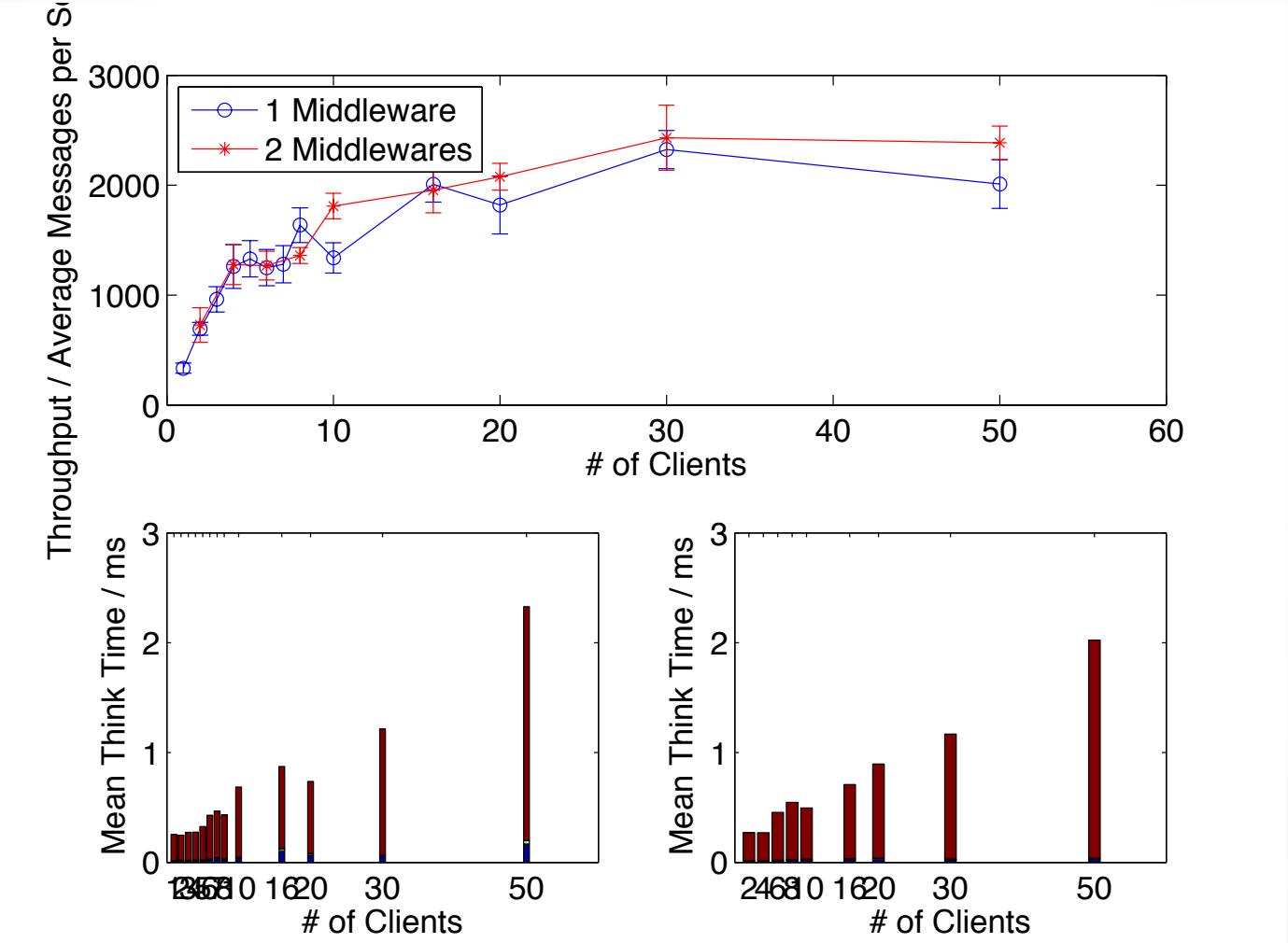
Results Of Experiments Conducted

Hot Spots - Method	Self time [%]	Self time	Invocations
org.postgresql.core.VisibleBufferedInputStream.**readMore** (int)		654,372 ms (61.6%)	173,020
org.postgresql.core.PGStream.**flush** ()		198,085 ms (18.7%)	173,020
asl.Persistence.PostgresPersistence.**executeQuery** (String, ja...		76,721 ms (7.2%)	72,243
asl.ThorBangMQServer.**read** (java.nio.channels.SelectionKey)		31,399 ms (3%)	72,404
asl.ThorBangMQServer.**send** (java.nio.channels.SelectionKey, S...		26,855 ms (2.5%)	72,323
asl.ThorBangMQServer.**write** (java.nio.channels.SelectionKey)		15,059 ms (1.4%)	72,323
org.postgresql.ds.jdbc23.AbstractJdbc23PoolingDataSource.**getP**		8,269 ms (0.8%)	86,467
asl.Persistence.PostgresPersistence.**executeStatement** (Stri...		5,543 ms (0.5%)	14,224
asl.Persistence.PostgresPersistence.**close** (java.sql.ResultSet, ...		4,547 ms (0.4%)	86,467
asl.ClientRequestWorker.**interpreter** (String)		2,853 ms (0.3%)	72,323

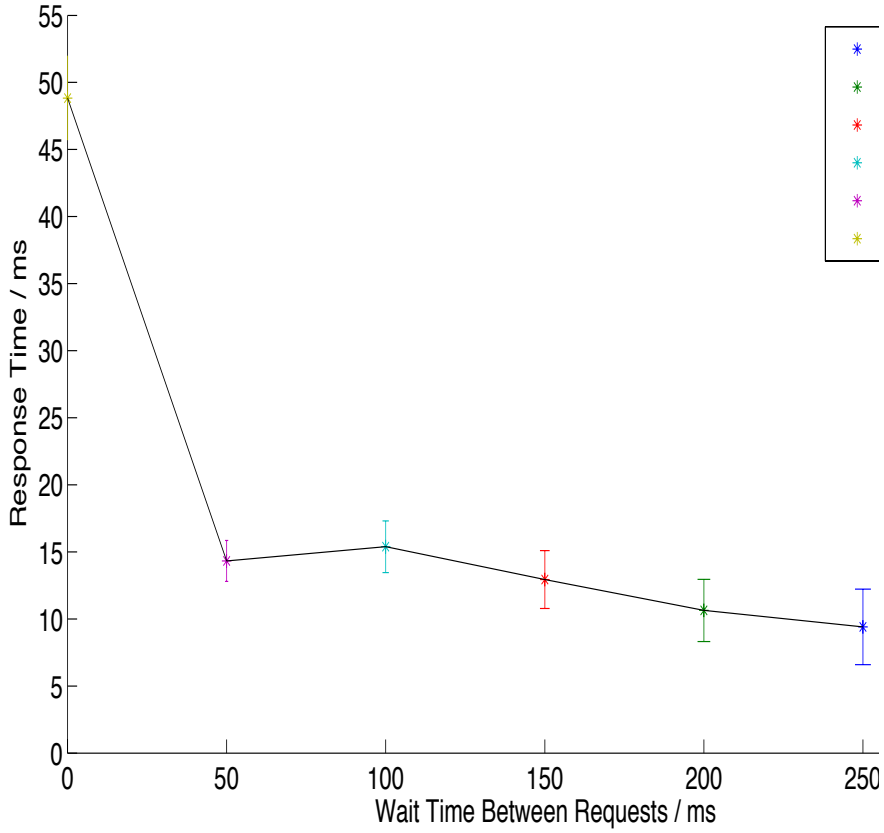


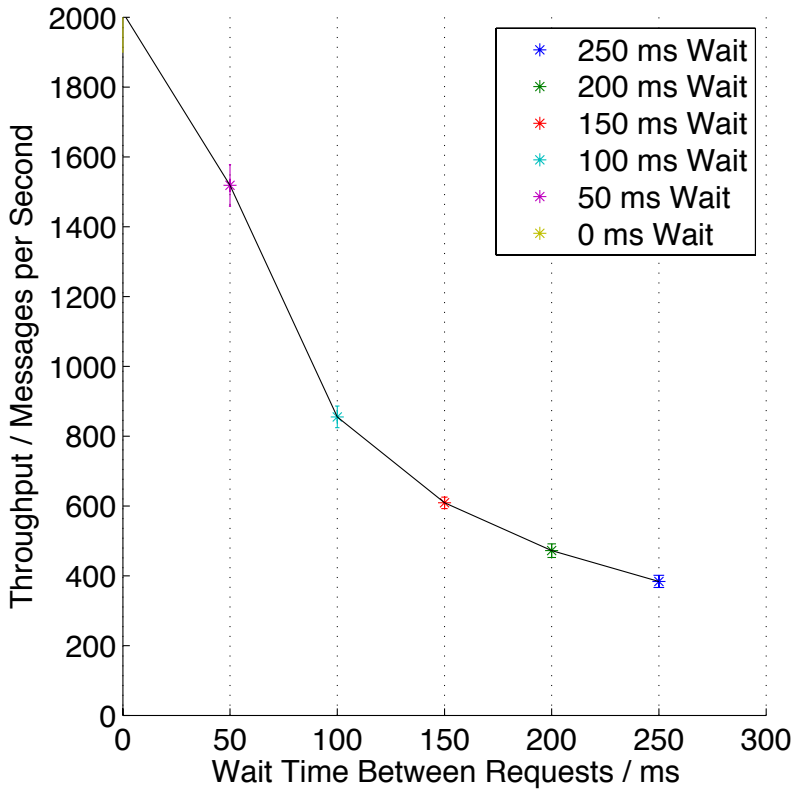
Top 3 methods make up 87.5% of processing time for middleware

Results Of Experiments Conducted



Results Of Experiments Conducted





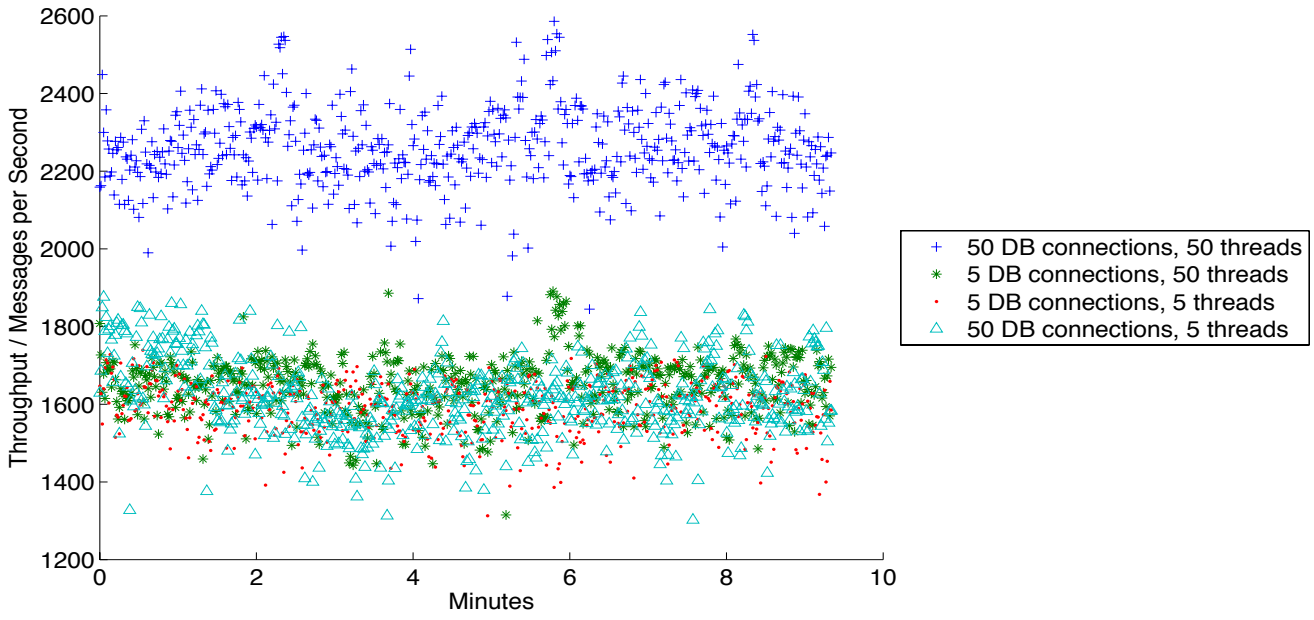
Results Of Experiments Conducted

Configuration	Mean Throughput	Response Time [ms]
10 Middlewares	2476.3 ± 2.2	1.15 ± 0.01
1 Middleware	2406.8 ± 19.1	44.55 ± 0.49
From 1 → 10	**+3% increase**	**30x decrease**

- 1000 one-way clients (total)
- 500 two-way clients (total)
- 30 db connections (per mw)
- 30 worker threads (per mw)

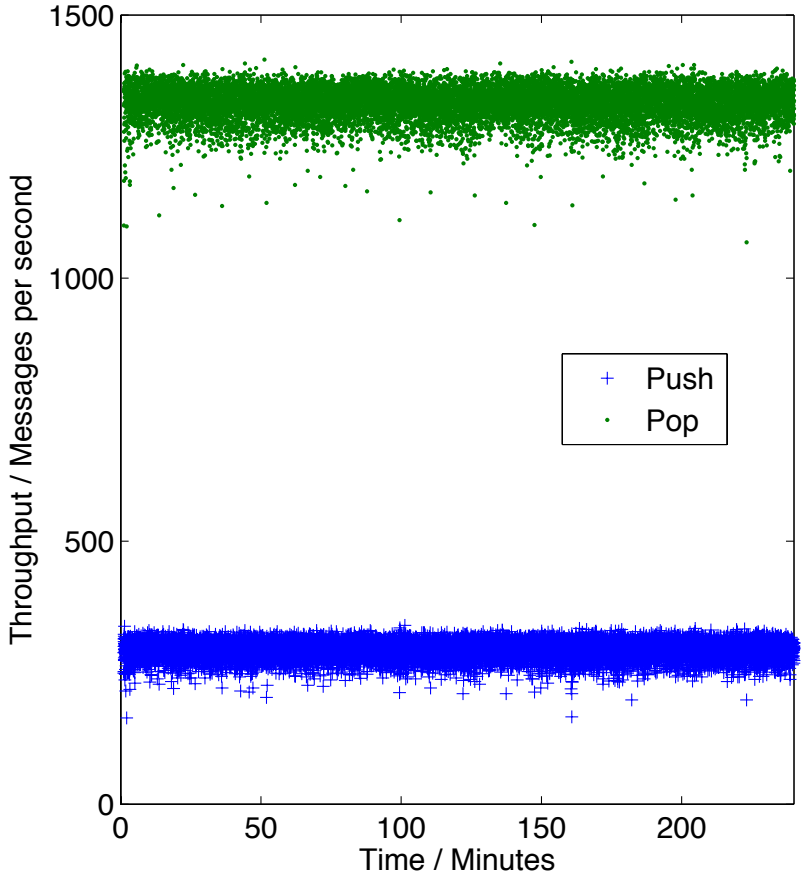
Results Of Experiments Conducted

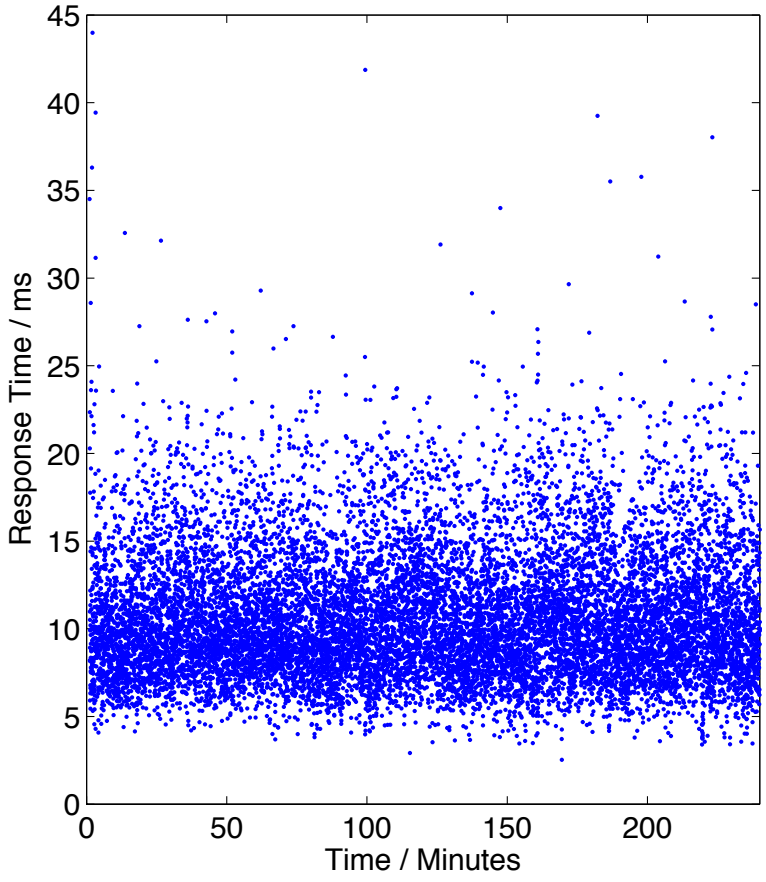
- Throughput is bounded by the minimum of the two parameters: worker threads and db-connections.
- With enough db-connections: performance in bouded by db



50 db-connections
50 worker threads
300 clients

4 Hour Trace





Mean push: 293.91 ± [-28.91 , 21.09]
Mean pop: 1331.73 ± [-53.73 , 42.27]
Mean resp. time: 10.66 ± [-4.54 , 7.06]

Things Learned

- Good Parameters
  - Worker Threads and db-connection 1:1 ratio
  - Worker Threads: ~40
  - When throughput hits 1500 consider adding middleware
- The database is the bottleneck
  - Thus increasing middlewares won't increasethroughput
- Maximum Throughput
  - Standard Test: ~ 2800 req/s
  - Send And Pop: ~2000 req/s
  - Bounded by the database