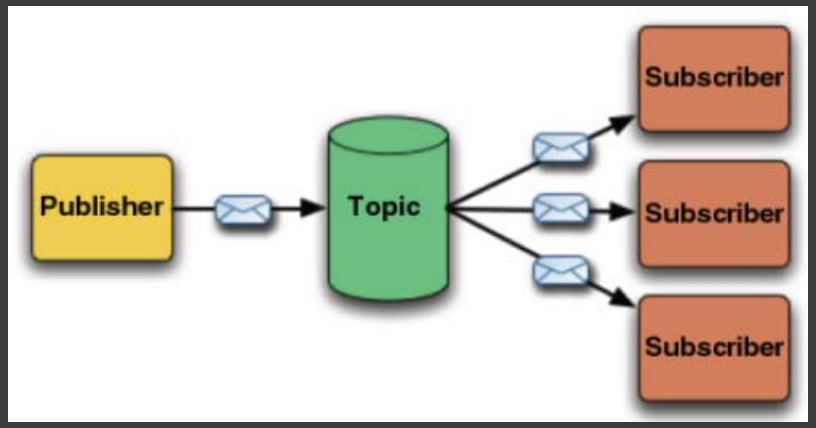
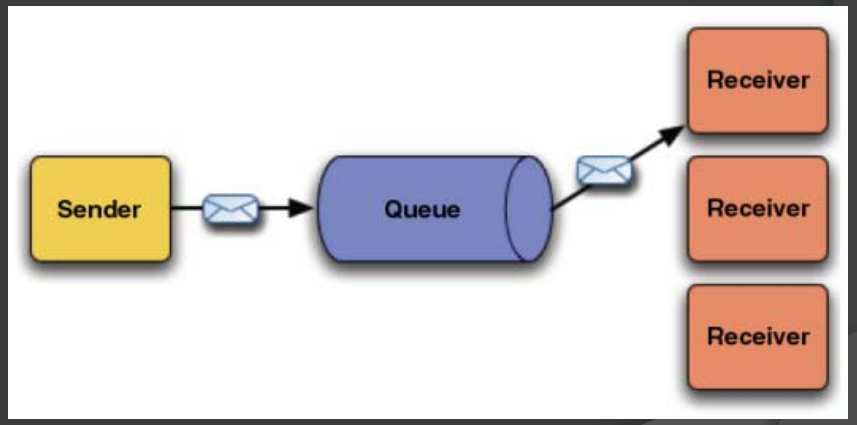
Status Report on JMS

- Introduction
- Deployment & Service Situation
- Tests & Results
- Future plans
- Conclusion

Introduction

- Java Messaging Service
  - Messaging API
  - Decouples Source and Destination
  - Topic / Queue concept





JMS Usage in BE-CO

- SonicMQ since 2000
  - LASER
  - DIAMON
- FUSE ActiveMQ since 2005
  - Projects
    - SIS, INCA, Sequencer, CESAR, PM/XPOC, Concentrators, Logging
  - Middle tier servers update GUIs
  - Middle tier to Middle tier

Current Deployment

- 2 SonicMQ Brokers
  - Deployed on 2 physical machines as cluster
- 10 ActiveMQ Brokers
  - 6 Production , 4 Development
  - Deployed on 6 physical machines
  - No cluster

Deployment Issues

Issues February 2009	June 2009
Production & Development share hardware	✘
Configuration not under version control	SOLVED
No test scenarios & tools	SOLVED
No unified way to upgrade / downgrade Broker	SOLVED
Insufficient Monitoring	SOLVED

- Monitoring
  - DIAMON
  - Problem recognition time 3min
  - Notification in case of problems via mail / sms

Service Issues

Issues February 2009	June 2009
Broker seems alive, but clinical dead	~
Only little knowledge about limits / behavior	SOLVED
No availability statistics	SOLVED

Current Status

- Availability (last month)

ping

Determined via sending test message

Broker	Host Avail.	Check Avail.
JMS-INCA-PRO	100%	99.998%
JMS-CO-PRO	100%	-
JMS-PUBLIC-PRO	100%	100%
JMS-SEQ-PRO	100%	99,577%
JMS-SIS-PRO	100%	99,529%
JMS-CESAR-PRO	99,975%	99,830%

Stress Test Setup

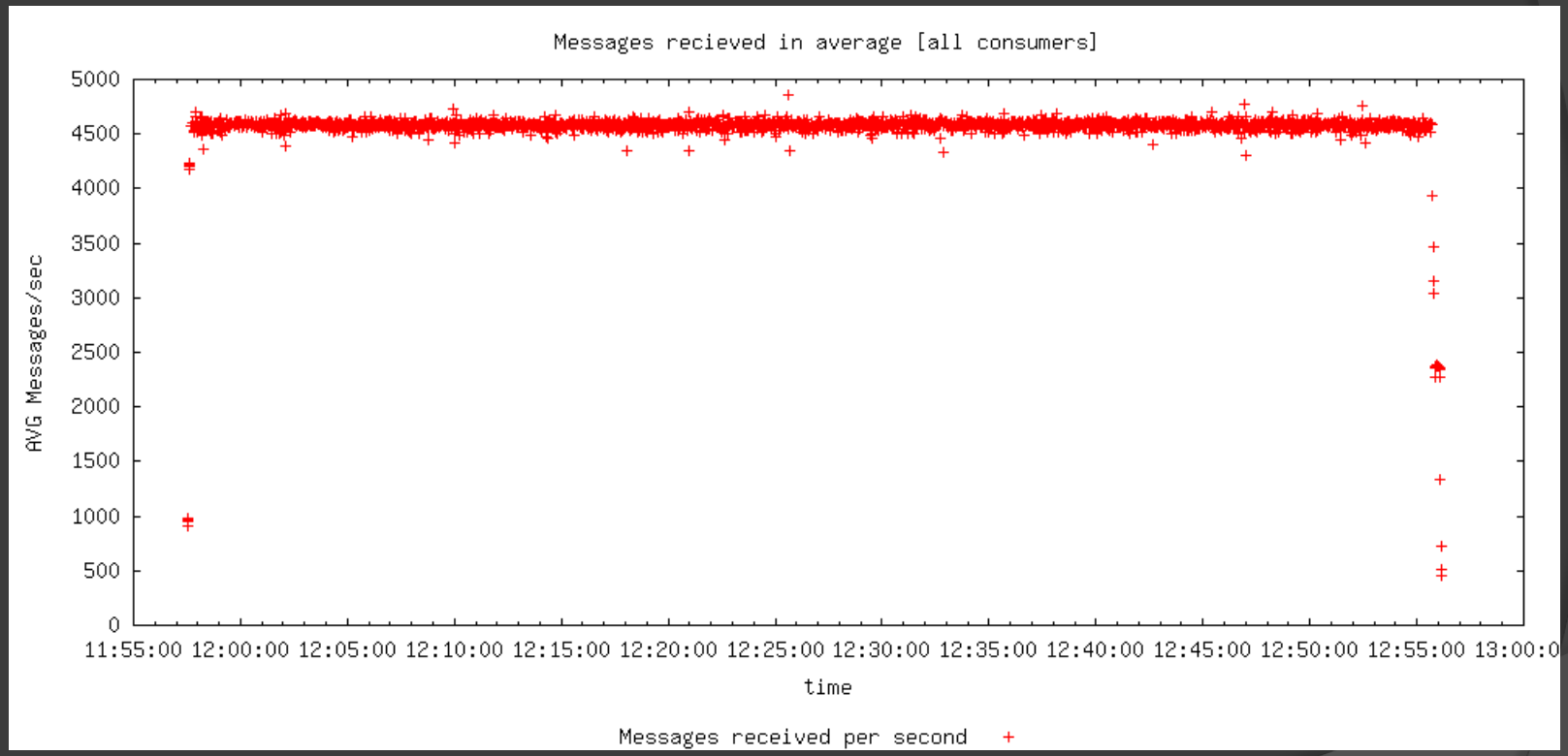
- 4 core machine with ActiveMQ 5.3 Broker
- > 5000 topics
- Message size 100Byte, timeout ½ h
- 40 Consumers
  - Each subscribes to all topics
- 6 Producers
  - Each publishes to all topics
- 300ms delay limit

Testing

- Default scenario
  - 4500 msg/sec, 5K Topics, 20K Subscriptions (500 Subscriptions / Connection)
- Measurement Aspects
  - Relation CPU load and Message throughput
  - Identify impact of
    - increasing Connections
    - increasing Subscriptions
  - Message delivery time

Result Example

- Messages received/sec over time

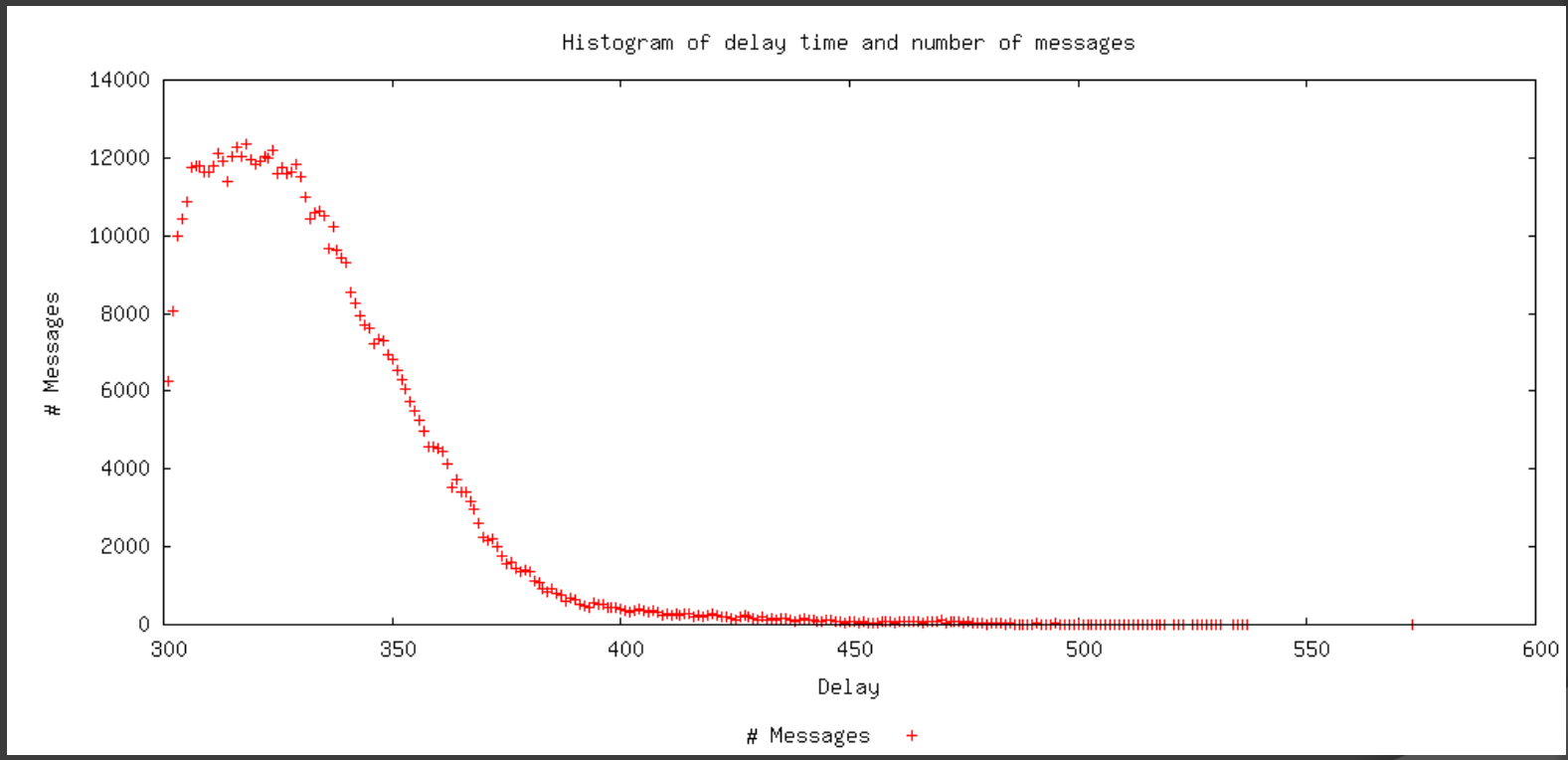


AVG Messages / sec

time

Result Example

- Histogram of delayed message distribution

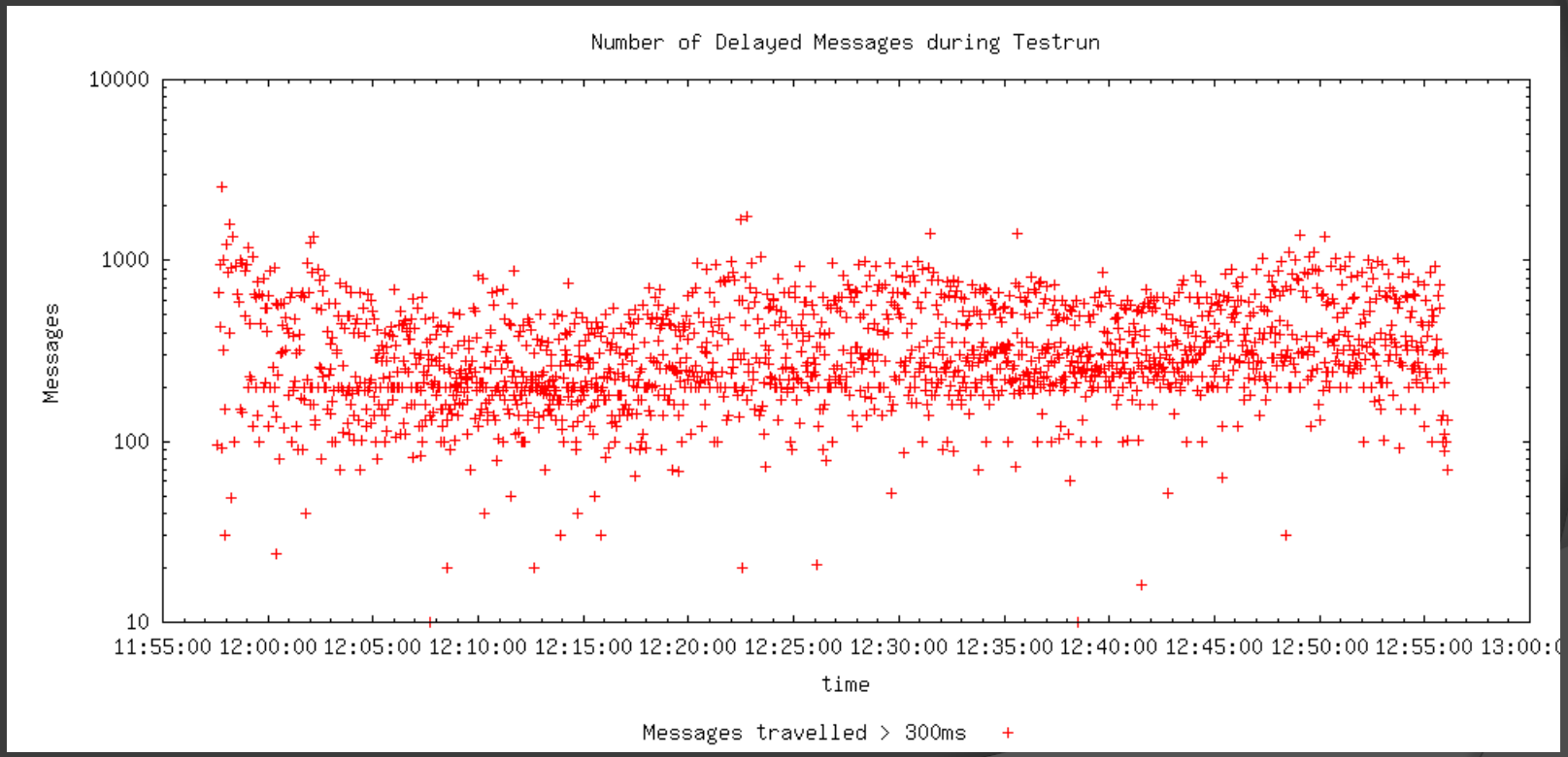


 # Messages

Delay time [msec]

Result Example

- Number of “late” (>300ms) messages for test duration



Delayed Messages

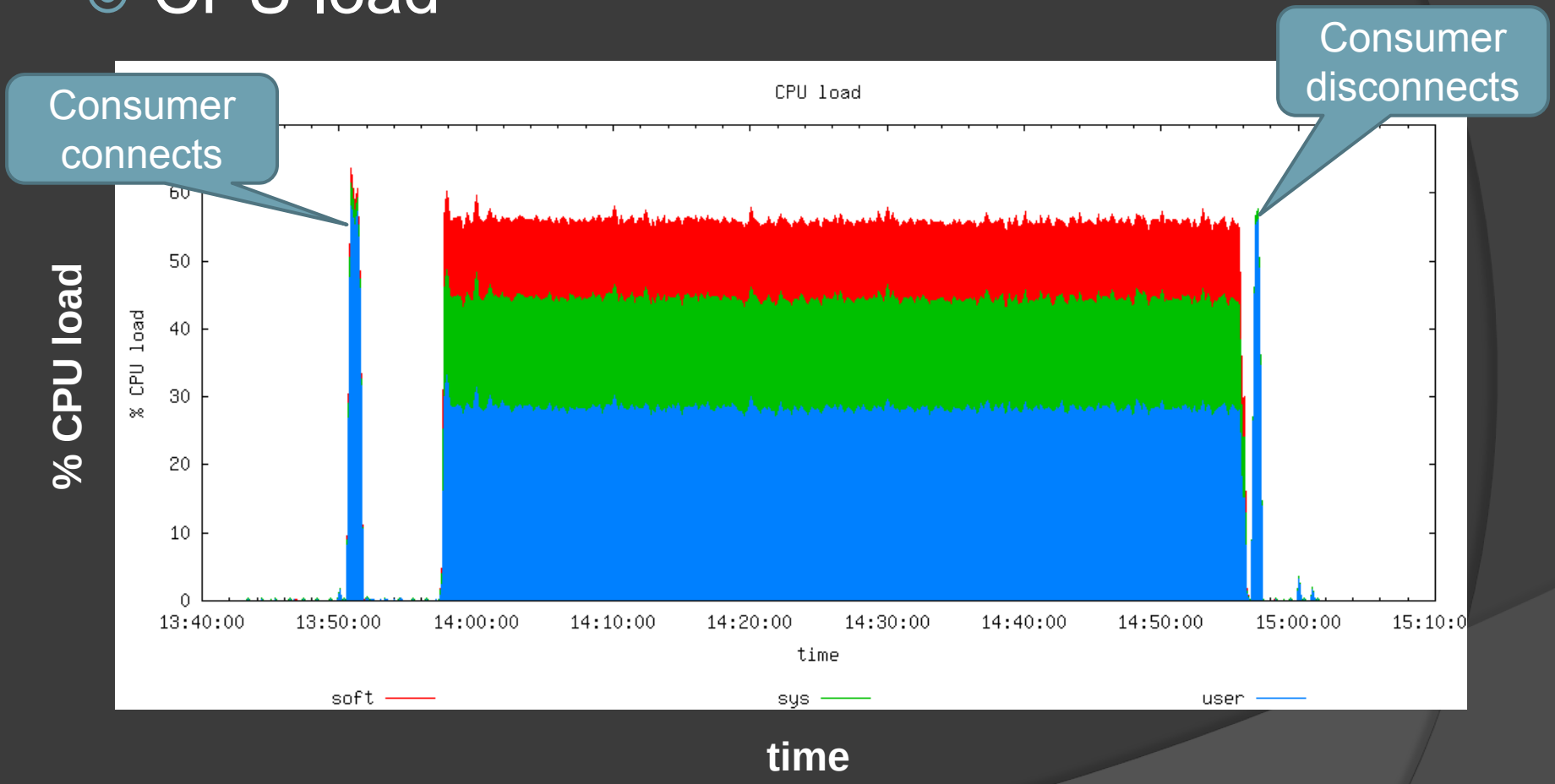
time

Message Delay Summary

- Measuring delayed Message
  - ~ 99.32% of messages delivered < 300ms
  - Rest < 1 sec
- Doubling numbers of messages (~9500 msg/sec)
  - 100% between 5 and 10 seconds
  - Further studies for improvement needed

Result Example

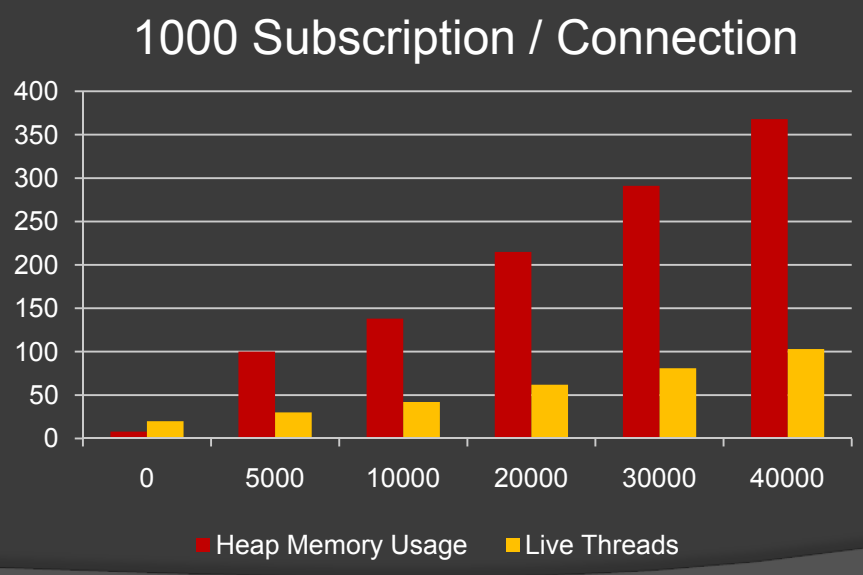
- CPU load

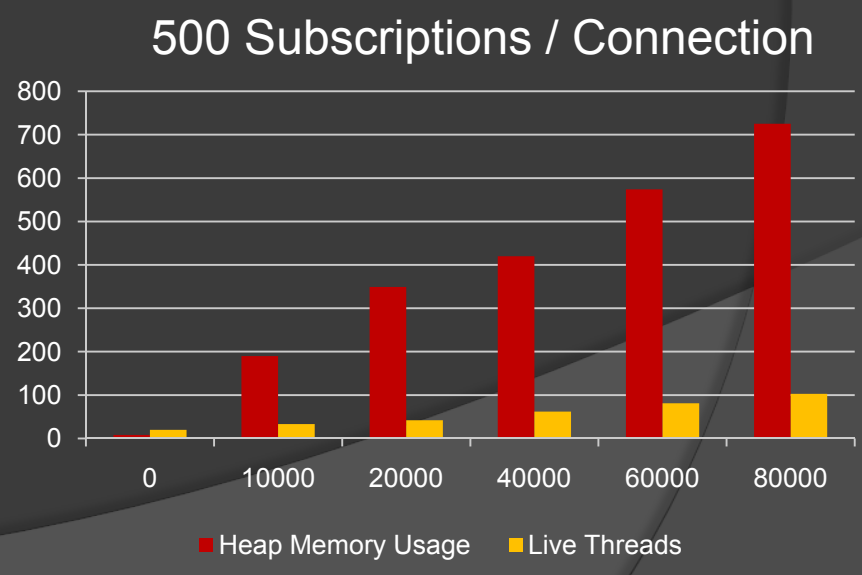


40% less CPU load thanks to optimization

Broker Memory Consumption

- Initial Memory is proportional to
  - Connections
  - Subscriptions





Persistence

- Messages are swapped to disk
- Shifts Java Memory Problem to larger scale

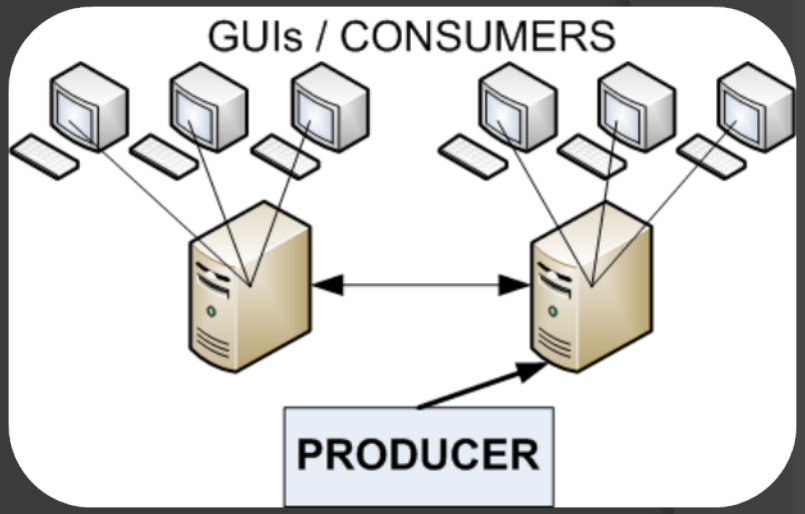
- Drawbacks
  - 1 Topic = 1 Persistence Thread
  - Increase of message delay
  - Higher CPU load

Future

- Persistence Tests
- Improvement of DIAMON integration
- Cluster Tests

Clustering I

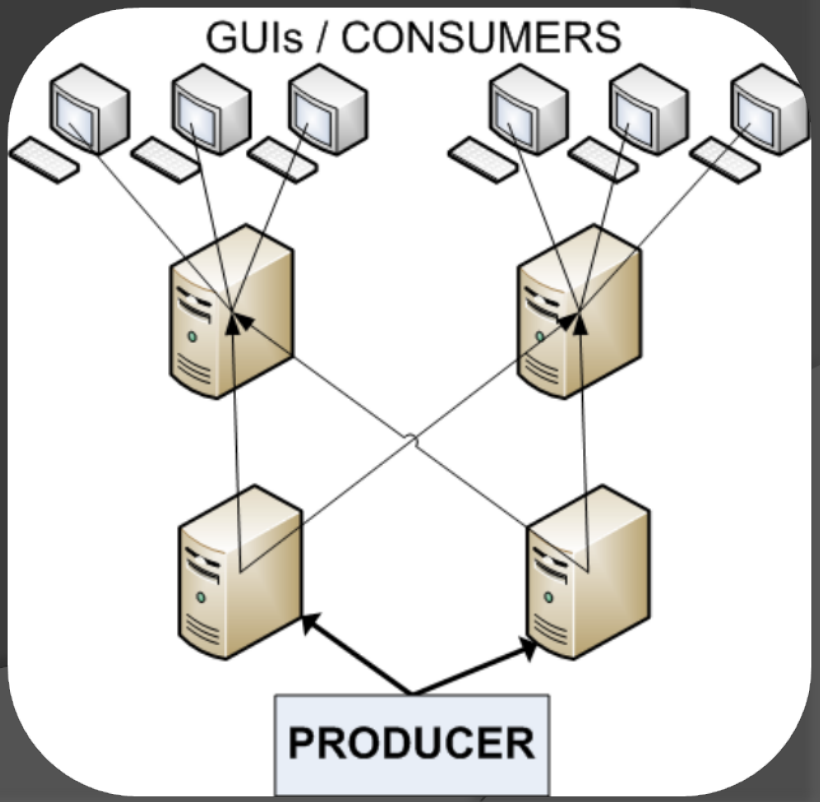
- Distribute Load
  - Interconnect two Brokers
  - Consumers/Producer can choose any Broker
  - Sharing of work



- Drawbacks
  - Consumer might not get all messages in case of network error
  - Routing overhead

Clustering II

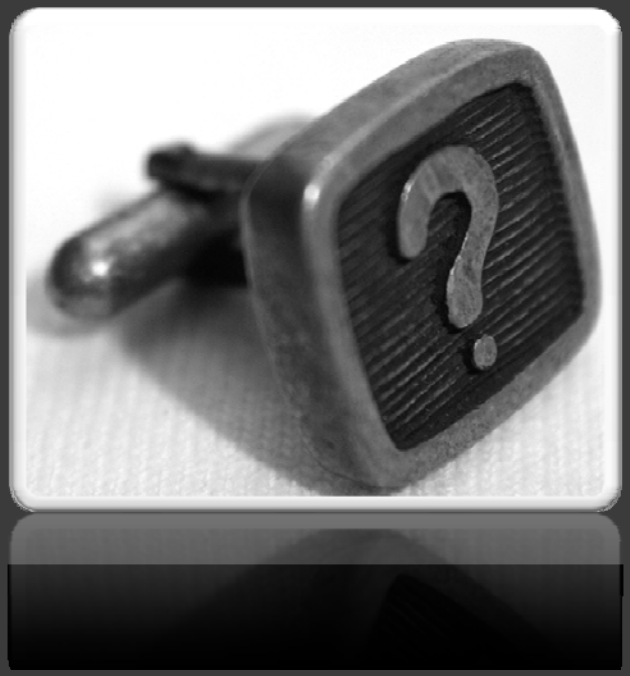
- Failover System
  - Message is sent to multiple Brokers
  - Consumer can get Message from any Broker
  - No routing overhead
- Drawbacks
  - Double sending message



Conclusions

- Official manpower for JMS (50%)
- Problems recognized and partly solved
- Providing proactive service
- Experience gained through tests
- Plans : DIAMON, Clustering, Persistence

Questions



References

http://wikis/display/MW/JMS