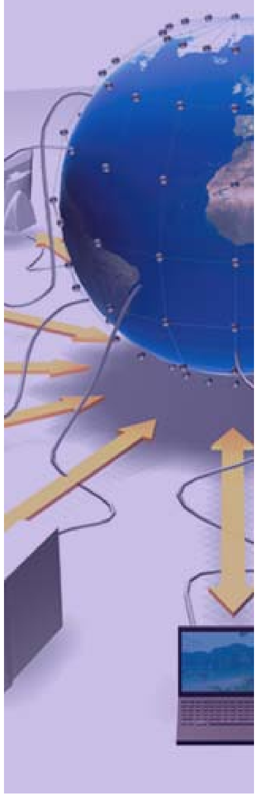


MSG: An Overview of a Messaging System for the Grid

Daniel Rodrigues

Presentation Summary

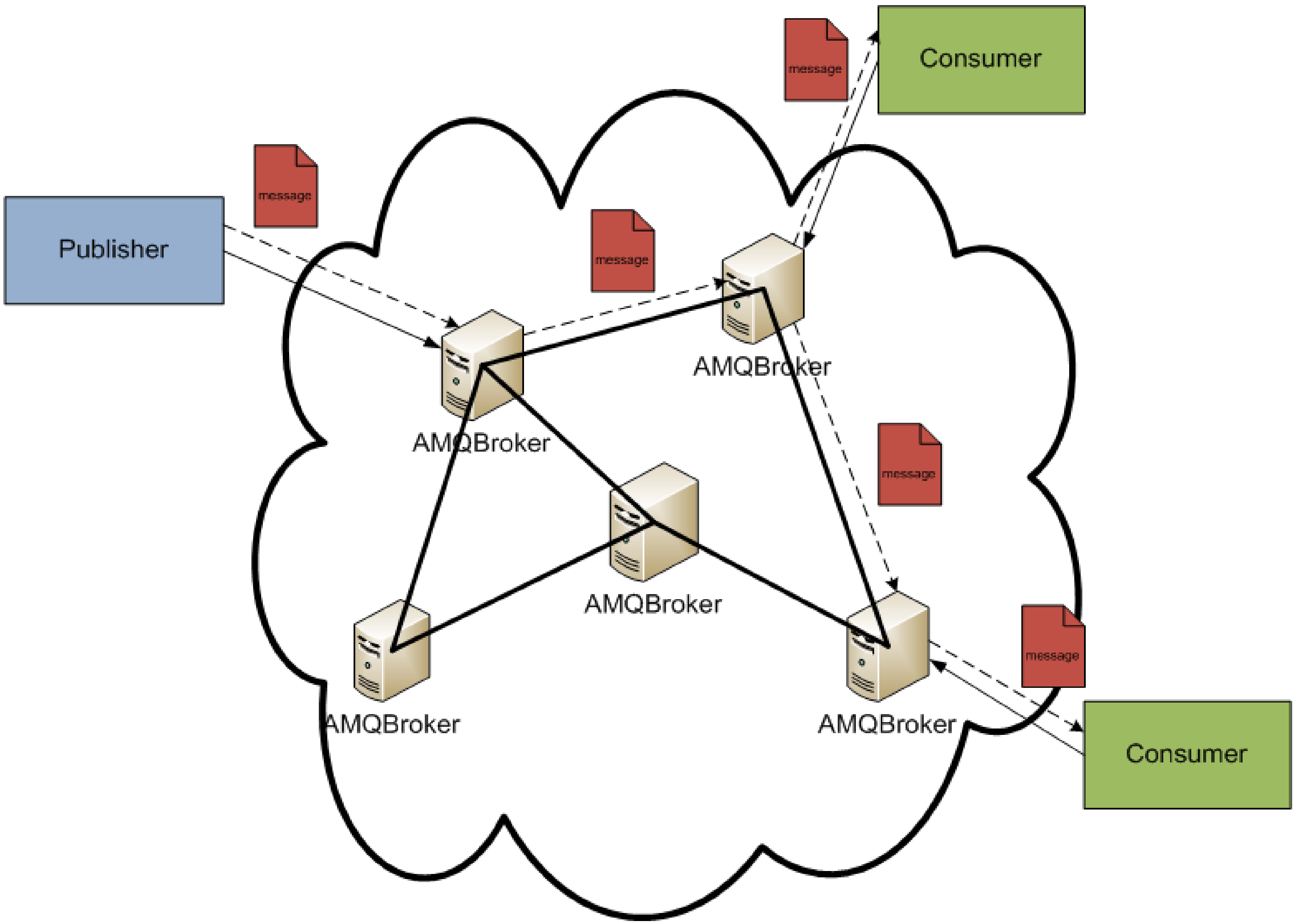
- Current Issues
- Messaging System
- Testing
- Test Summary
  - Throughput
  - Message Lag
  - Flow Control
- Next Steps

Current Issues

- The current paradigm in the Grid is based on “Distributed central services”
- Single points of failure exist within Grid Monitoring Systems
  - (ex: Service Availability Monitoring [SAM]).
- Reliability on information delivery is often not guaranteed…
- … as is not Scalability.

- Will a Messaging System improve both reliability and scalability?

Messaging System



Testing

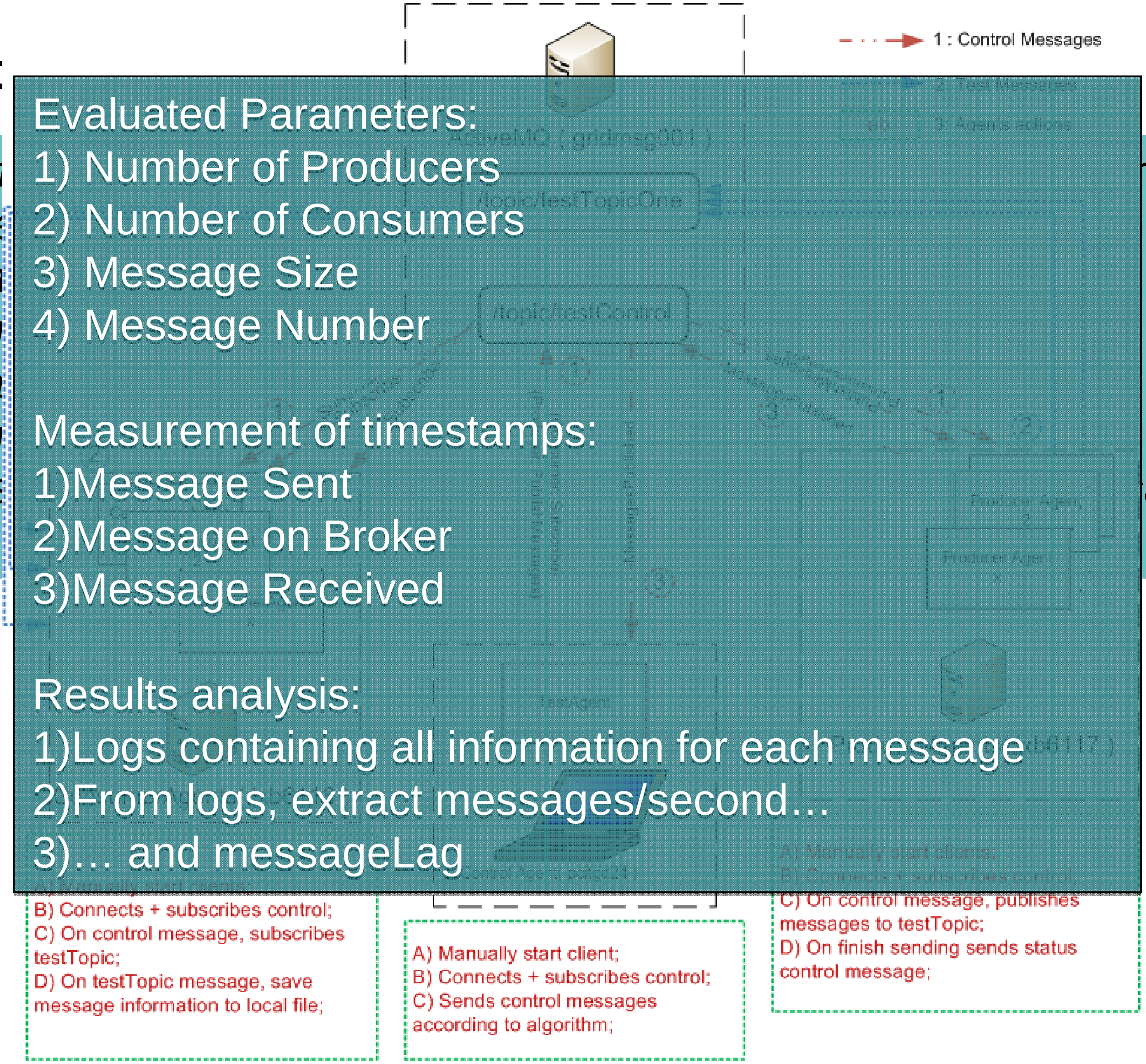
- F

Per

- the
- tra
- qu
- ha
- nu
- dis

s:

age size



Evaluated Parameters:
1) Number of Producers
2) Number of Consumers
3) Message Size
4) Message Number

Measurement of timestamps:
1)Message Sent
2)Message on Broker
3)Message Received

Results analysis:
1)Logs containing all information for each message
2)From logs, extract messages/second…
3)… and messageLag

A) Manually start clients;
B) Connects + subscribes control;
C) On control message, subscribes testTopic;
D) On testTopic message, save message information to local file;

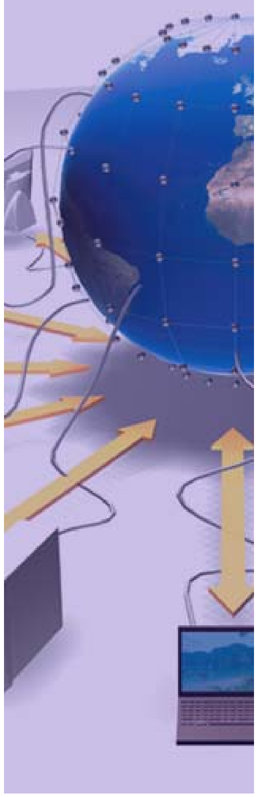
A) Manually start client;
B) Connects + subscribes control;
C) Sends control messages according to algorithm;

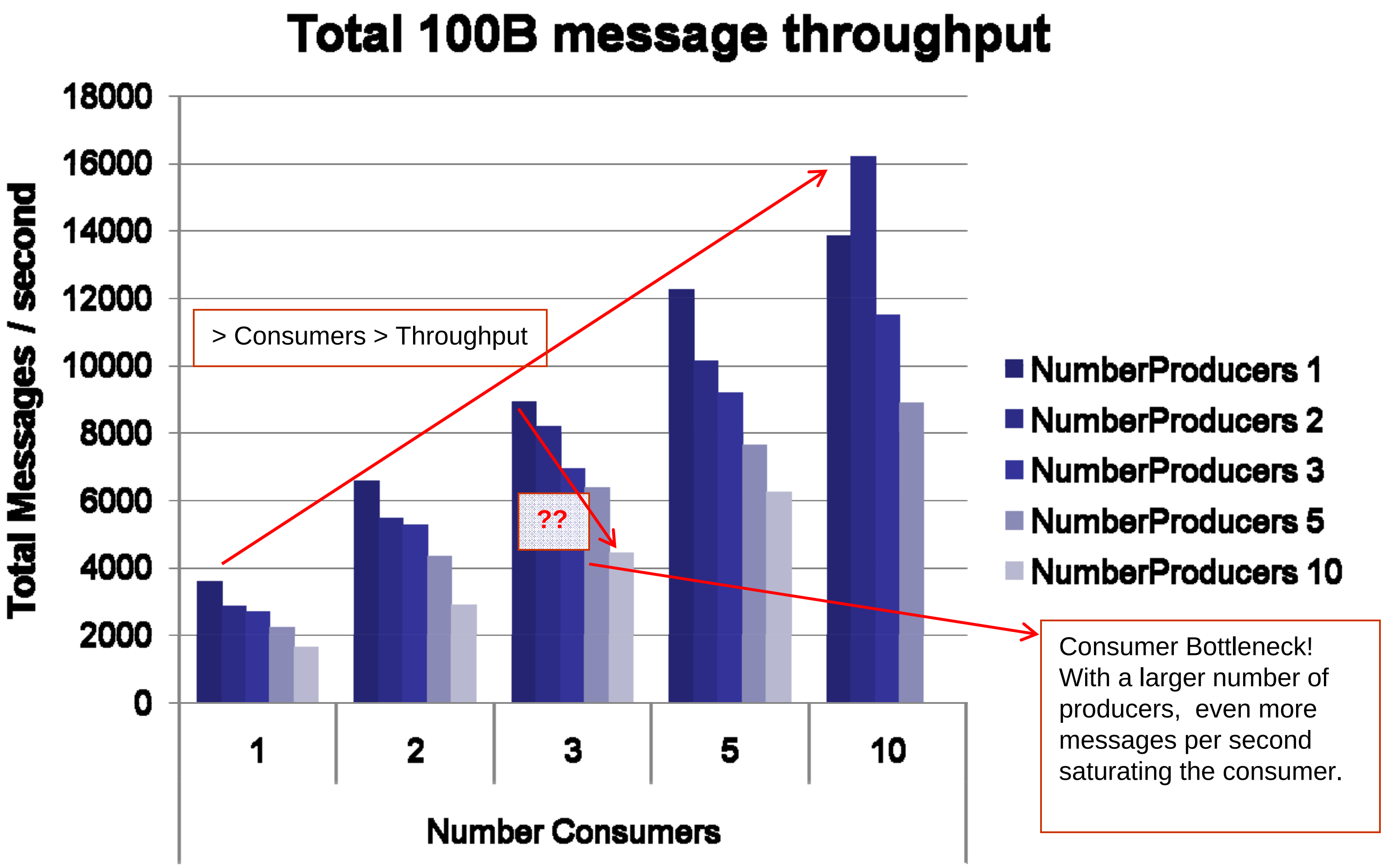
A) Manually start clients;
B) Connects + subscribes control;
C) On control message, publishes messages to testTopic;
D) On finish sending sends status control message;

Test Summary

- Broker statistics:
  - Running for 6 weeks with no crashes
  - 50 Million messages of various sizes (0 to 10 kB) forwarded to consumers
  - 12 Million incoming messages from producers
  - Up to 40 Producers and 80 Consumers connected at the same time
  - Stable under highly irregular test pattern:
    - Number of clients change
    - Frequent client process kills
    - Daily number of tests vary

Results : Throughput



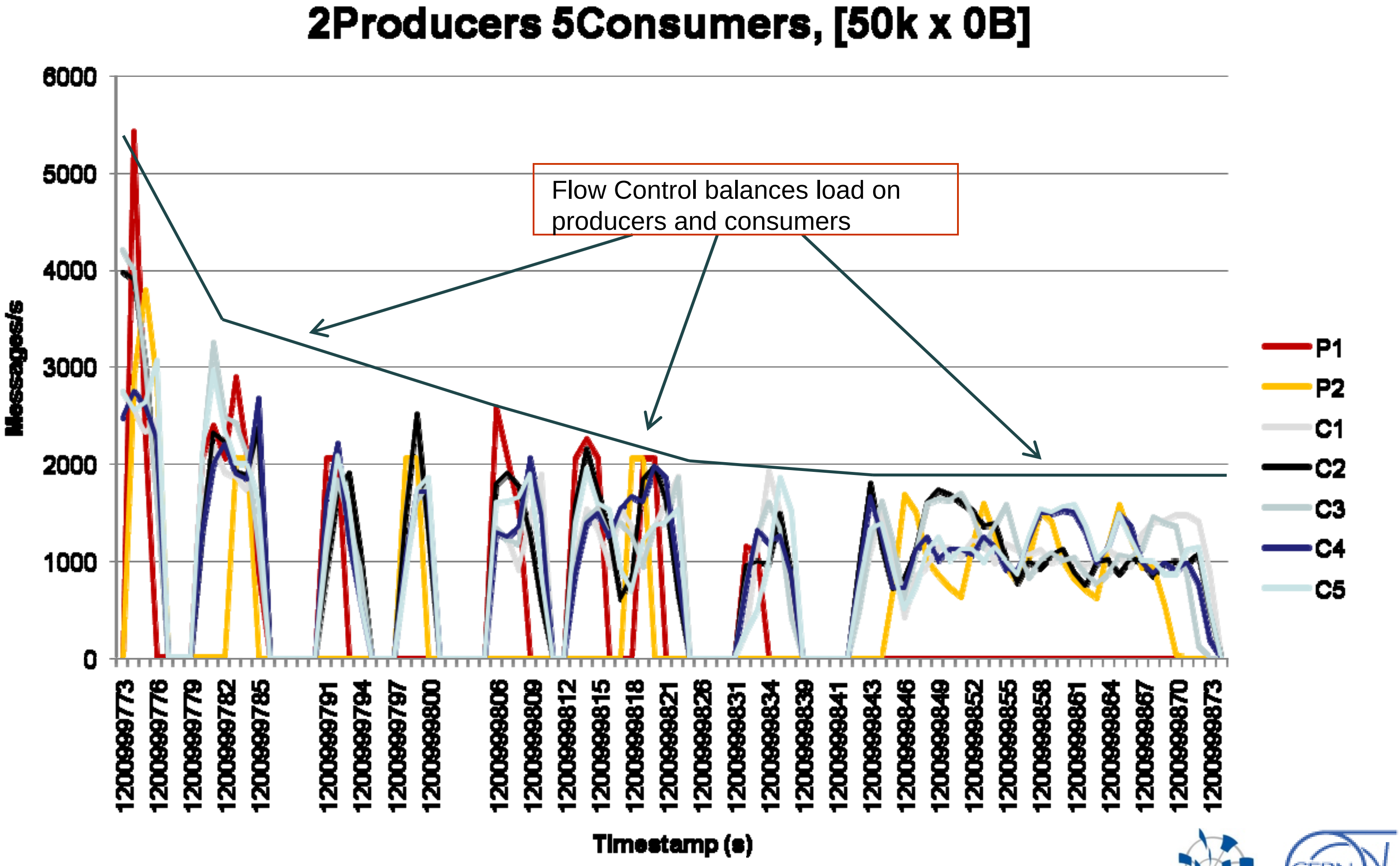


> Consumers > Throughput

Consumer Bottleneck!
With a larger number of producers, even more messages per second saturating the consumer.

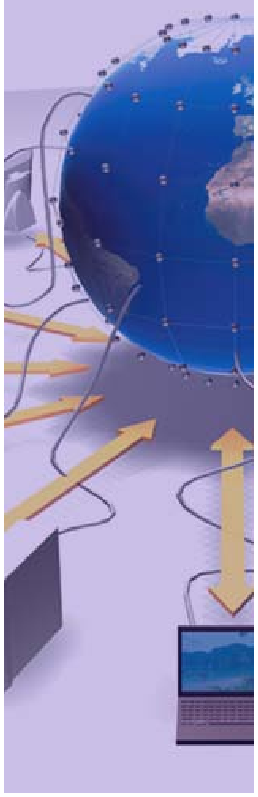
Results : Flow Control

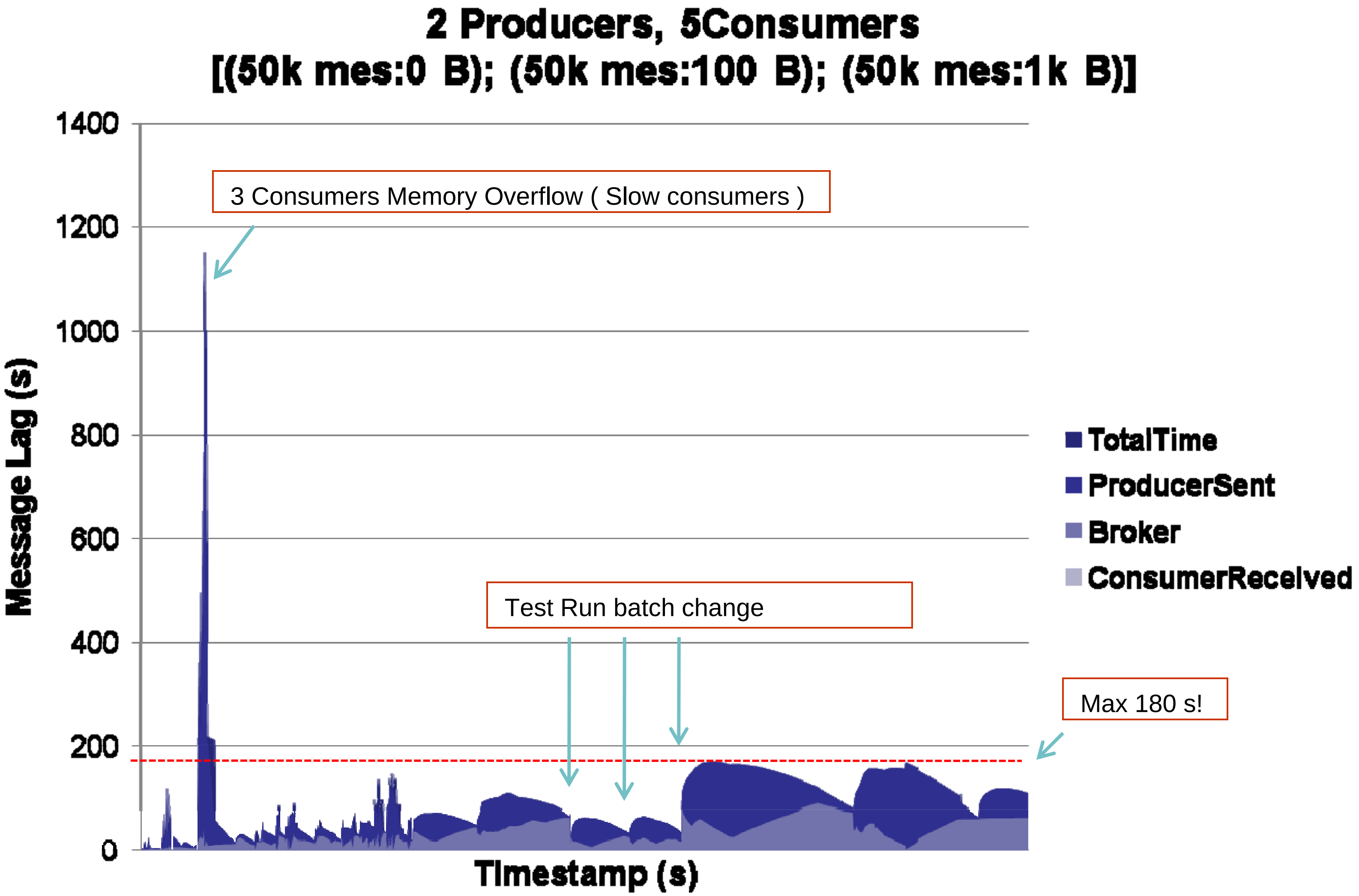
- Effective flow control



Flow Control balances load on producers and consumers

Results : Message Lag





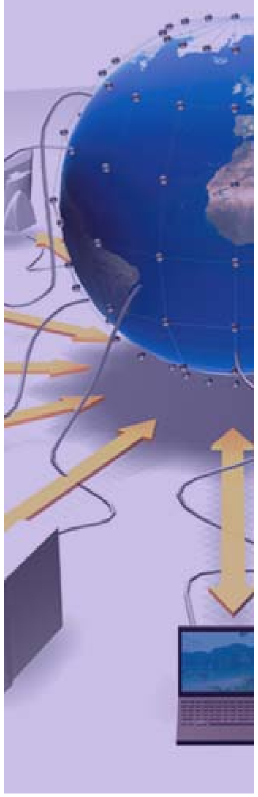
3 Consumers Memory Overflow (Slow consumers)

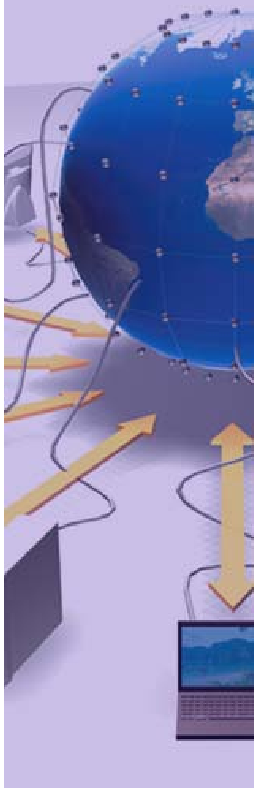
Test Run batch change

Max 180 s!

Next Steps

- Scalability in a distributed environment
  - Network of Brokers
  - Testing optimized wire protocols (OpenWire)
- Evaluation under real world use cases
  - SAM
    - 1 Consumer ~ 300 Producers per VO
    - 15 (~2k) messages / second
    - Prototype already in place for OSG
  - Atlas
    - 10 Producers ~ 100 Consumers
    - Streaming of messages with 200 B each
    - Persistence required

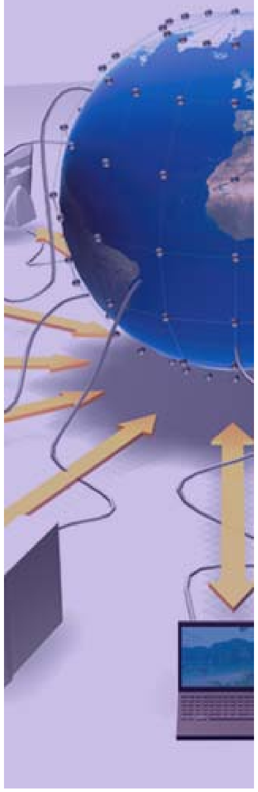




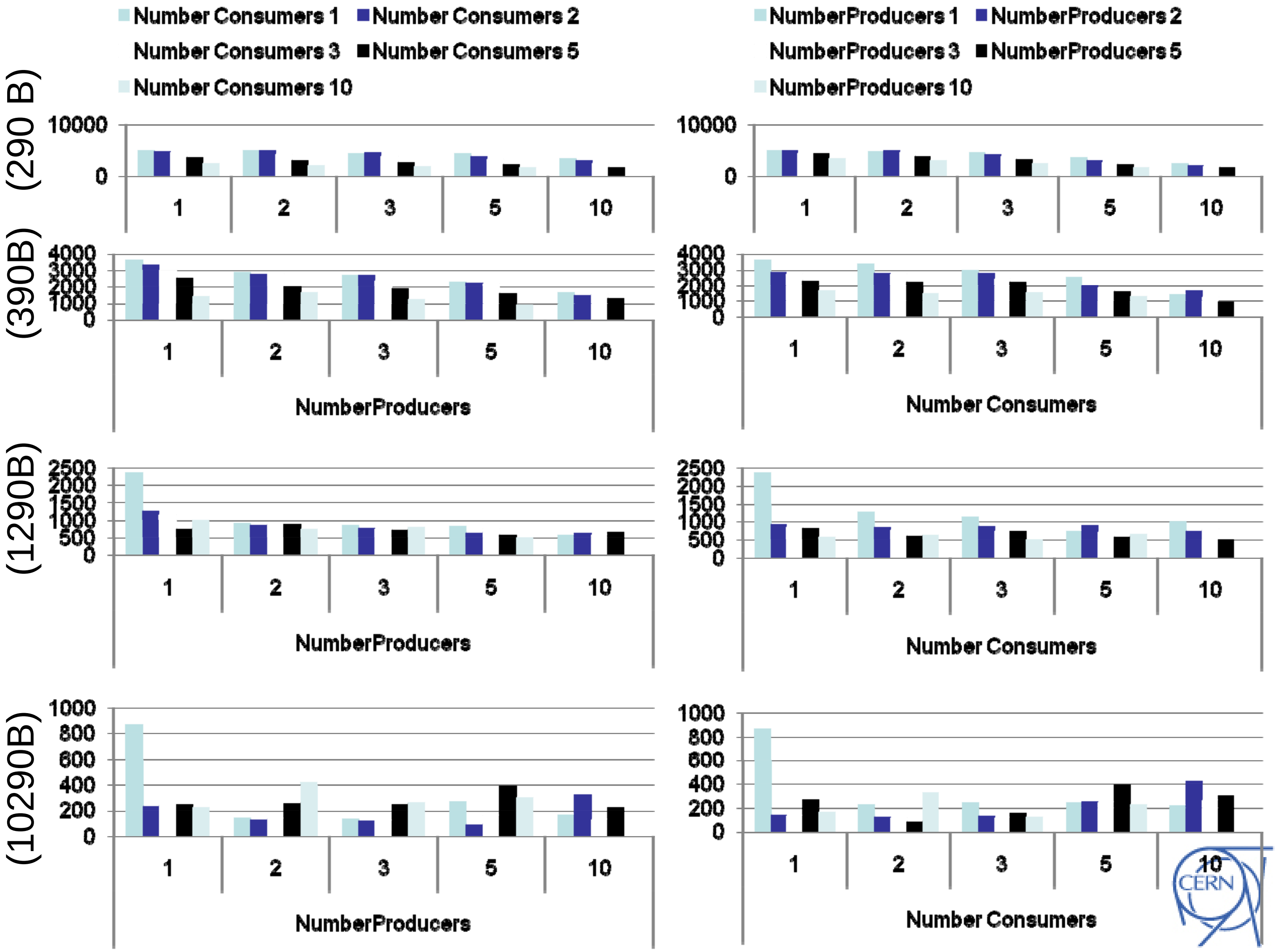
Thank you for your attention.

GS

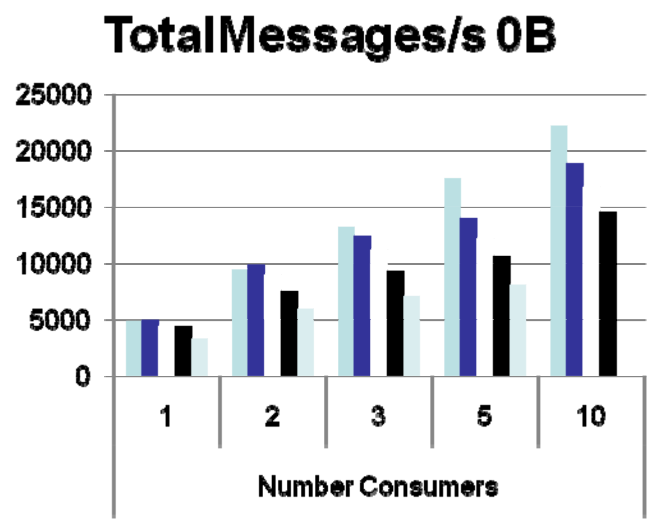
Support Slides

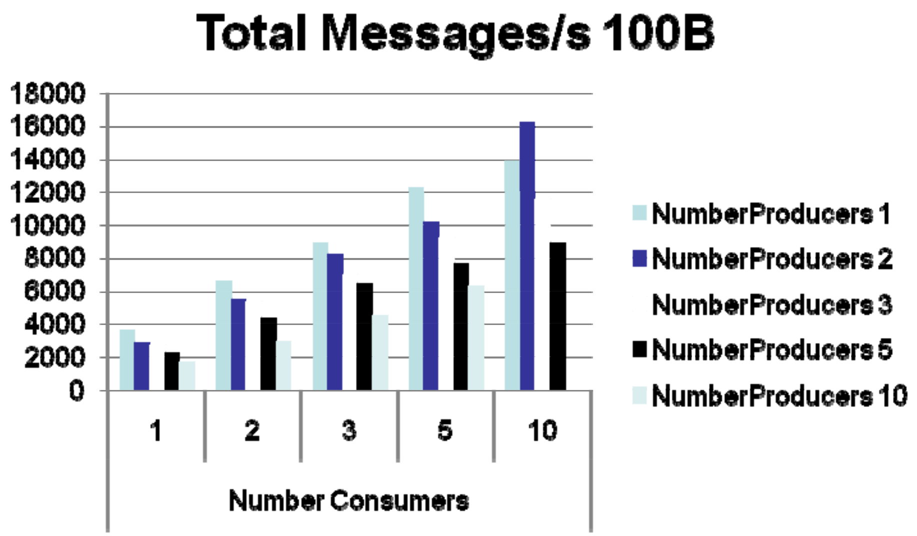


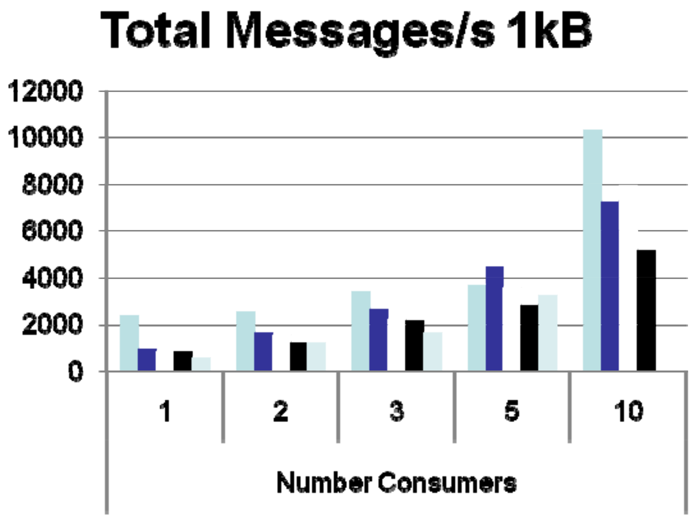
Messages/s / Consumer

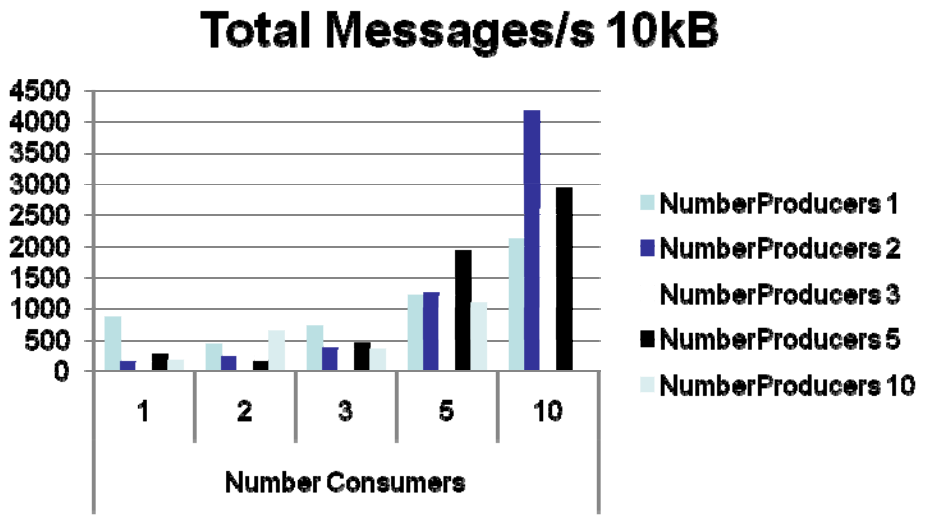


Total Messages/s

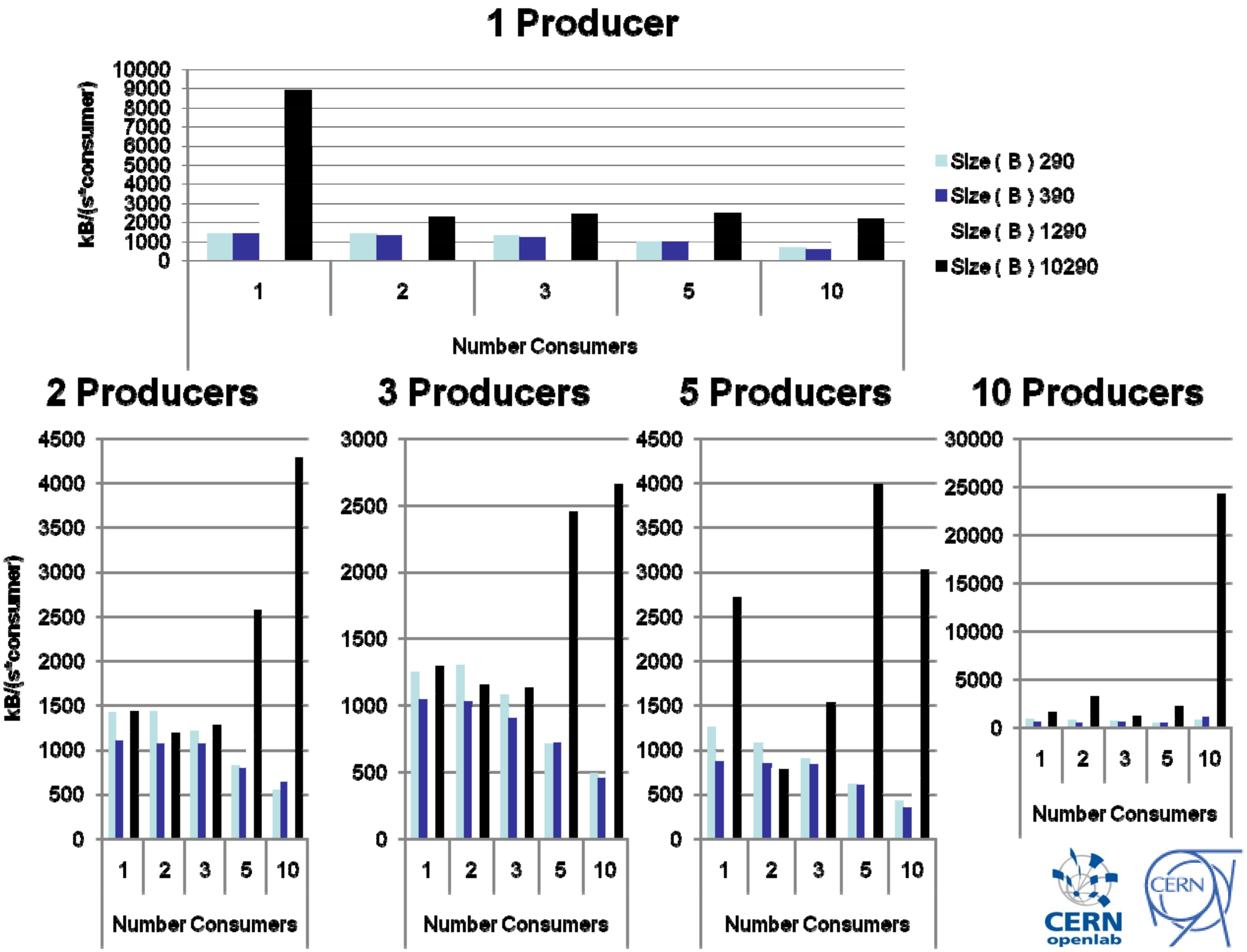

Total Messages/s 0B


Total Messages/s 100B

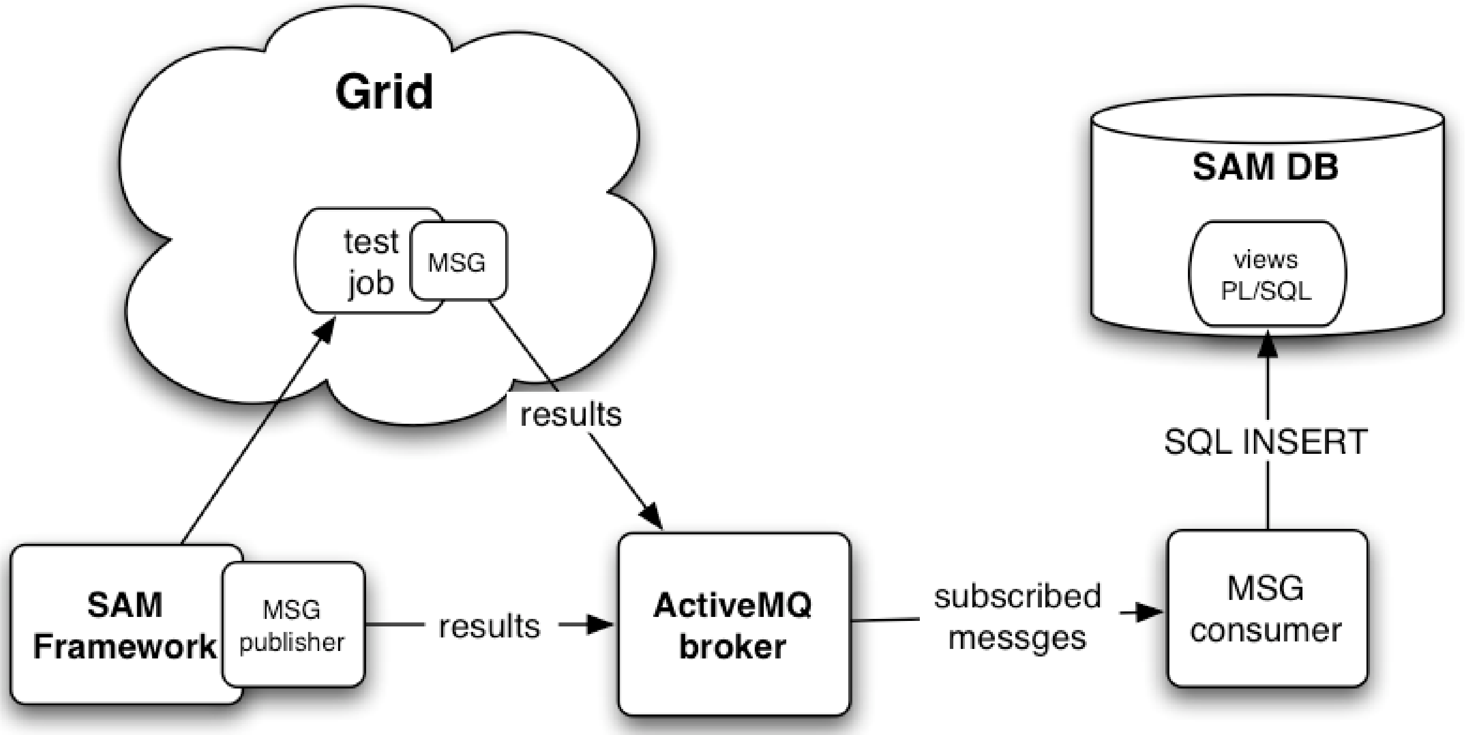

Total Messages/s 1kB


Total Messages/s 10kB

Kb/s per consumer

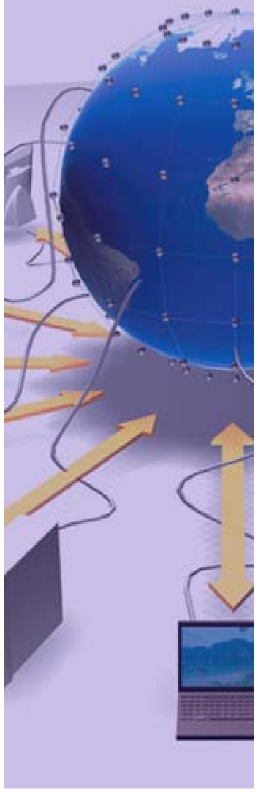


SAM-MSG integration overview (Piotr Nyczyk)



- Test results are published both from the framework and directly from test jobs executing in grid sites
- MSG-consumer is using “transport views” in Oracle DB (see later)

SAM – MSG: Publishing side



- Firewall and network issues - test jobs running on Worker Nodes
  - solution: using HTTP protocol (REST) with http_proxy if available
  - robust publisher: list of broker URLs (STOMP/REST), the first one that responds is used to publish
  - requirements: message servlet installed on the broker machine
- Tested with a typical SAM load for 1 VO
  - message rate: 1 to 10 messages/second
  - published from many short-lived producers
    - ~300 machines (producers) publishing at the same time
    - ~15 messages for each producer
  - prototype setup with 1 broker (gridmsg001)
- Currently used for OSG monitoring integration with SAM

SAM – MSG: Consumer side

- Generic consumer written in Python:
  - durable subscription (no data loss in case of producer downtime)
  - message classes based on WLCG MW Probe Format: key-value pairs
  - trivial transformation to SQL inserts:
    - message class - table (name mapping)
    - attribute (key) - column
- On the Oracle DB side:
  - a view for each message class with exactly the same columns as the attributes
  - PL/SQL code in “INSTEAD OF INSERT” trigger to do the ID look-ups and actual insert(s) into underlying tables