Imperial College London

Department of Computing

Living in the Present: On-the-fly Information Processing in Scalable Web Architectures

David Eyers, Tobias Freudenreich, Alessandro Margara, Sebastian Frischbier, **Peter Pietzuch**, Patrick Eugster

University of Otago, TU Darmstadt, Imperial College London, Purdue University

dme@cs.otago.ac.nz, freudenreich@dvs.tu-darmstadt.de, margara@elet.polimi.it, frischbier@dvs.tu-darmstadt.de, prp@doc.ic.ac.uk, p@cs.purdue.edu

CloudCP Workshop – April 2012

Importance of Social Web Platforms

Use of online social web platforms growing at staggering pace:

Twitter

- 11 new accounts are created per second
- More than 300 million users in 2011
- Over 2200 tweets and over 18,000 queries per second, spikes at up to 4× that load

Facebook

- Over 800 million active users and 100 billion hits per day

➔ Therefore their architectures are under strain

Real-Time Data Processing Platforms

Changing role of social web platforms (e.g. Facebook, Twitter, etc.)

- Once places just to collect and display digital artefacts

Rather than reporting on the world, social networks now actually shaping it directly!

- Use of Twitter in Arab uprising, and other protests globally
- … yet much of the analytics operates off-line using large batch jobs

Emerging role:

Processing large amounts of user-generated data **on-the-fly**

Sample Scenario: Location-based Advertising

Social networks are increasingly accessed using mobile devices

- Companies want to advertise services/products via social networks
- Potential customers should be targeted based on interests & location

Real-time location-based advertising

- Conversations on social platforms can be mined in real-time for terms that match advertised products/services
- Current geographical location of each customer (e.g. GPS on smartphone) correlates with advertised products/services nearby
- Customised ads are pushed to mobile devices when in proximity

Social web platforms such as Facebook allow third-party add-ons

- Place new real-time requirements on infrastructure

Main Idea

Time to rethink fundamentally the distributed architecture of social web platforms

- Focus on processing fresh data responsively
- Relegate storage-focused components to historical data management
- Exploit publish/subscribe communication for real-time data processing

Outline:

1. Evolution of social web platforms
2. Storage-centric platform model ➔ Publish/subscribe platform model
3. Open challenges and conclusions

Evolution of Social Web Platforms

Platforms have been changing architecture frequently

- Twitter launched July 2006: new memory cache layers needed by year 4
- Facebook: wide assortment of software platforms has accumulated

In particular, relational databases result in problems:

- Twitter added in-memory caches but…
- …dropped MySQL back-end: 10-20% service rejection during FIFA World Cup
- LinkedIn launched 2003: soon dropped Oracle/MySQL
- Facebook developed own infrastructure (Cassandra) to scale up

We believe: object stores are only half-way to ideal solution

- Push computation into request-handling part of network, not storage layer

Move Towards Real-time Processing

All sorts of custom systems have popped up:

Twitter	LinkedIn	Facebook
Lucene	Kafka (Scala +Zookeeper)	FB Messages: Epoll
Storm (CEP)		Historic: Cassandra

Analysis and web platform are typically still separate systems

- Facebook: Hadoop and Hive for offline processing (Hbase storage)
  - Also use Scribe and ScribeHDFS: logging & click-stream analysis
- Twitter Storm and Yahoo S4 for offline analysis of streams

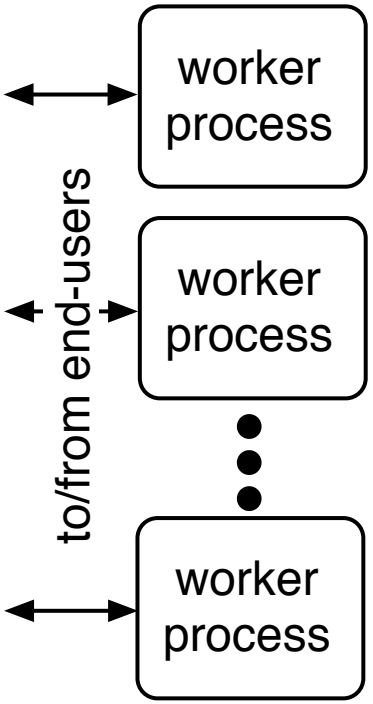
Core web presence still tends to be storage-centric

Storage-centric Architecture

Existing architecture usually has three main software layers

Worker processes

- Link end-user processes into social web platform
- Correlate stored information to present data to users



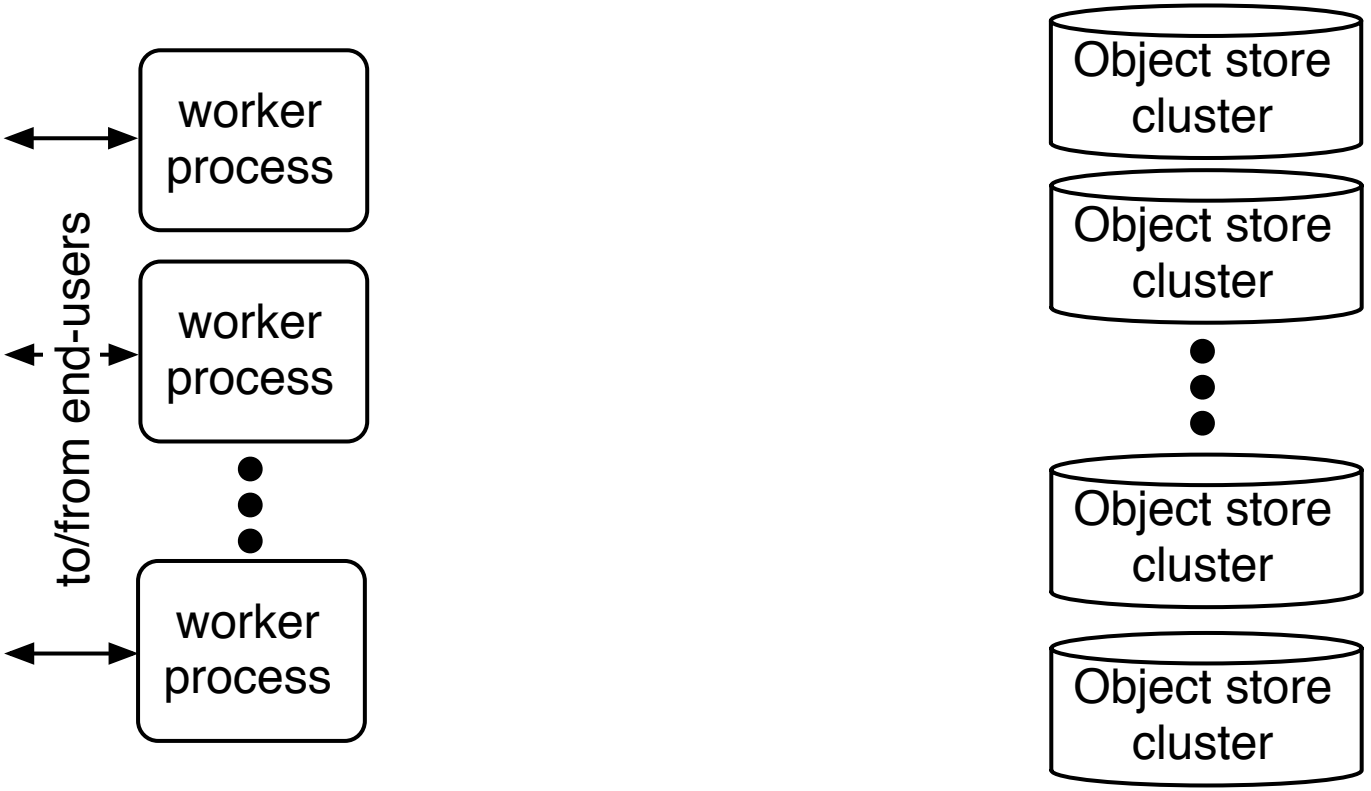
Storage-centric Architecture

Storage often done using NoSQL object stores

- Restricted expressiveness, e.g. no support for complex "join" operations

Object store distributed over cluster

- Better scalability than clustered relational databases



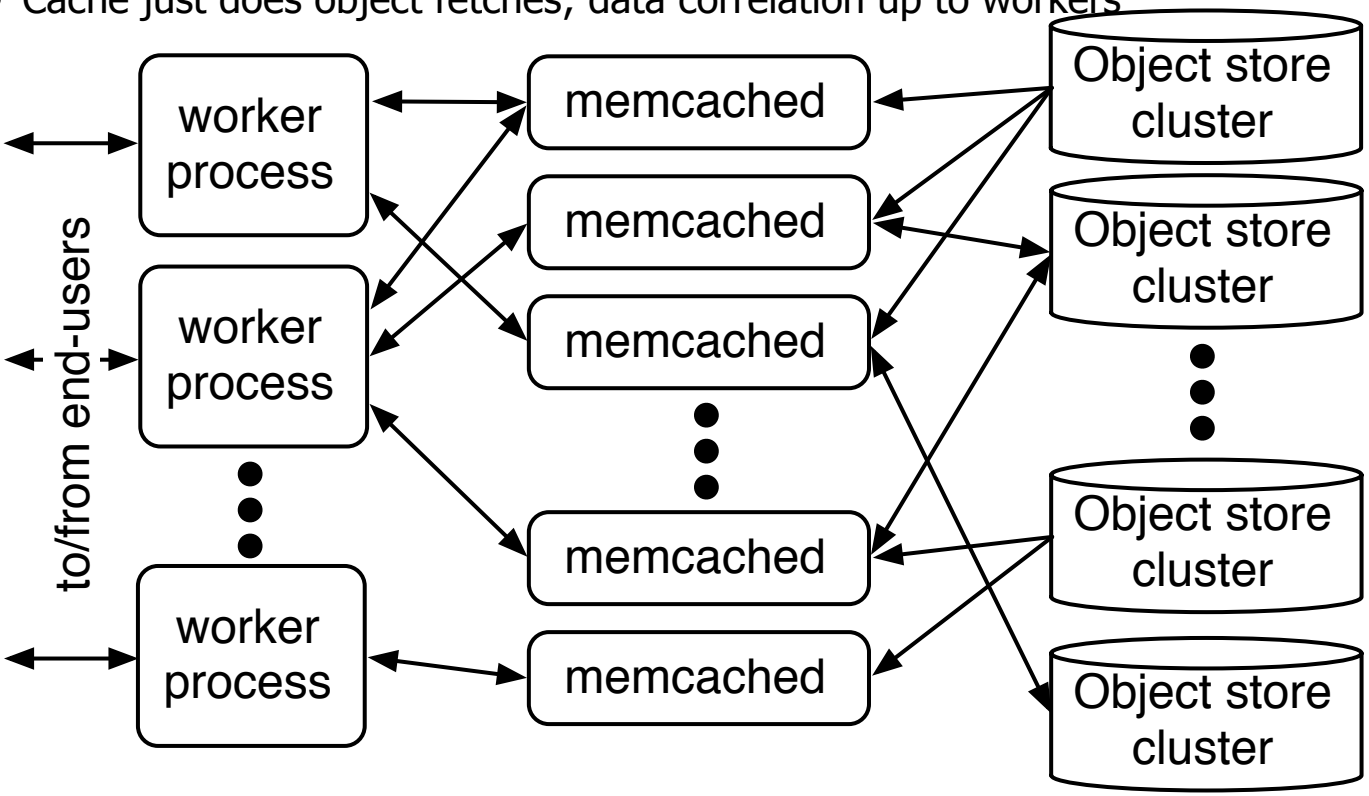
Storage-centric Architecture

Memory caching layers reduces I/O latency

- Often distributed over cluster (e.g. memcached)

Key problems

- **Semantic mismatch** between cache and store
- Not a **push architecture** for updates
  - Cache just does object fetches; data correlation up to workers



Future Evolution of Storage-centric Architecture

Main message:
”Architecture of social web platforms should be around live communication and not storage”

Use unified design for querying, analysing & storing data

- Unlike storage-centric: not just caching data items
  - Cache has semantic awareness, captures data interconnections & dependencies

Support for inherently push-based updates

- Simplifies platform work in providing timely interface to users
- Strengthens consistency (Facebook frequently returns stale data)

Exploit publish/subscribe communication paradigm…

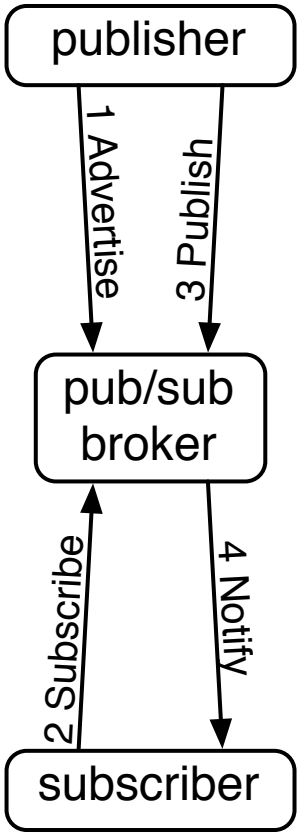
Publish/subscribe Communication

Publish/subscribe paradigm:

- Connects publishers (senders) and subscribers (receivers)
- Uses topics or message content (instead of explicit destination addresses)

Message Brokers manage interconnection:

1. Publisher advertises intent to publish
2. Subscriber indicates topics/message content of interest
3. Publishers publish messages agnostic to subscribers
4. Subscribers are notified of matching messages



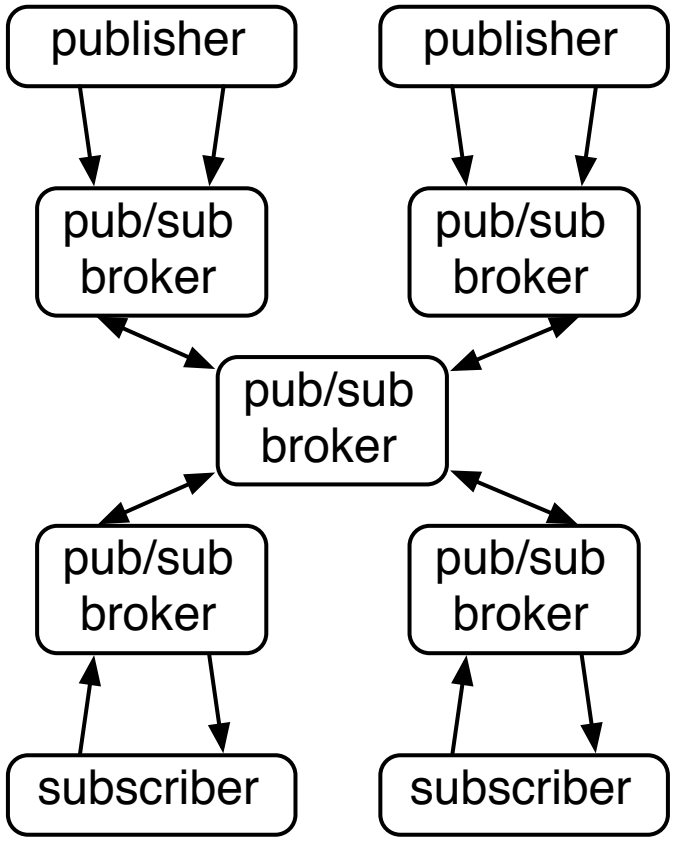
Distributed Publish/subscribe

Publish/subscribe communication with multiple message brokers

- Makes communication infrastructure more scalable and resilient
- Message dissemination graph formed across brokers
- Spanning tree connects pubs/subs

Brokers form message processing network

- Perform computation at brokers on the path of messages
- Allows direct processing of message data in transit



Publish/subscribe Architecture

Key point:
Perform data processing within broker network

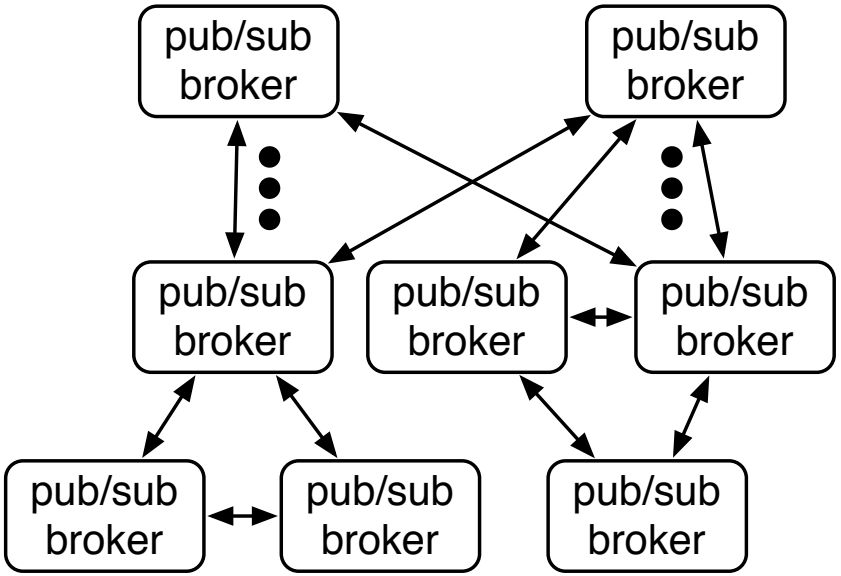
- Merge cache and object-store layers

Brokers take responsibility for data

- E.g. subscriptions to posts with "platypus" tag

Broker topology matches data centre network hierarchy

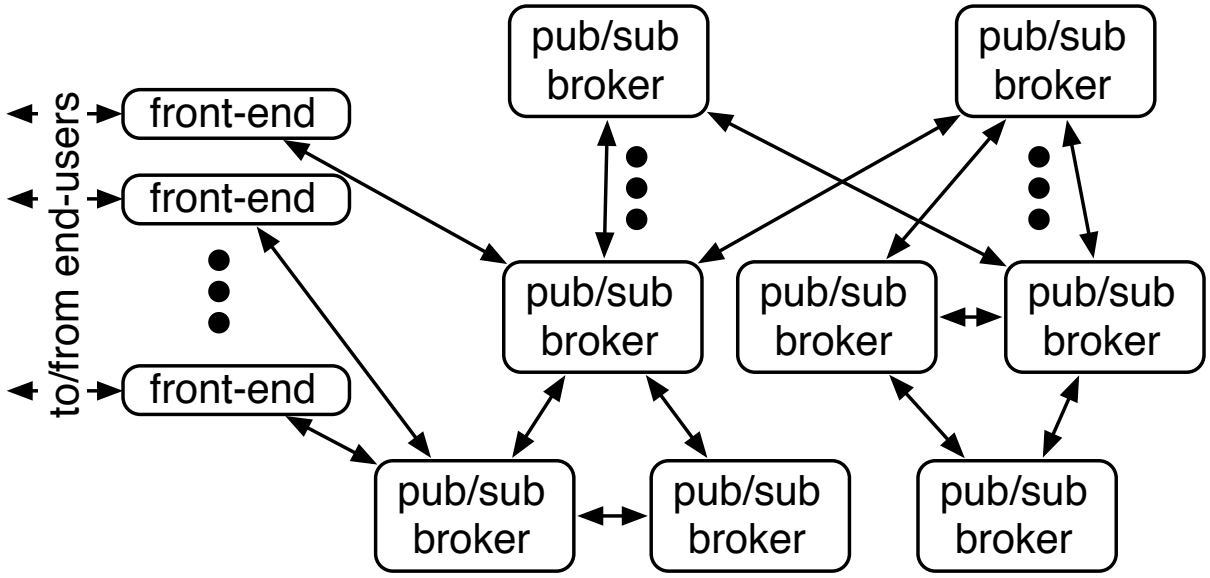
- Extra inter-broker links increase resilience to network failures



Publish/subscribe Architecture

Offload computation from front-end worker processes

- Front-end processes become subscribers and publishers in publish/ subscribe back-end
  - Directly facilitates push-updates to front-end results
- Front-end should ideally only format and serialise user requests

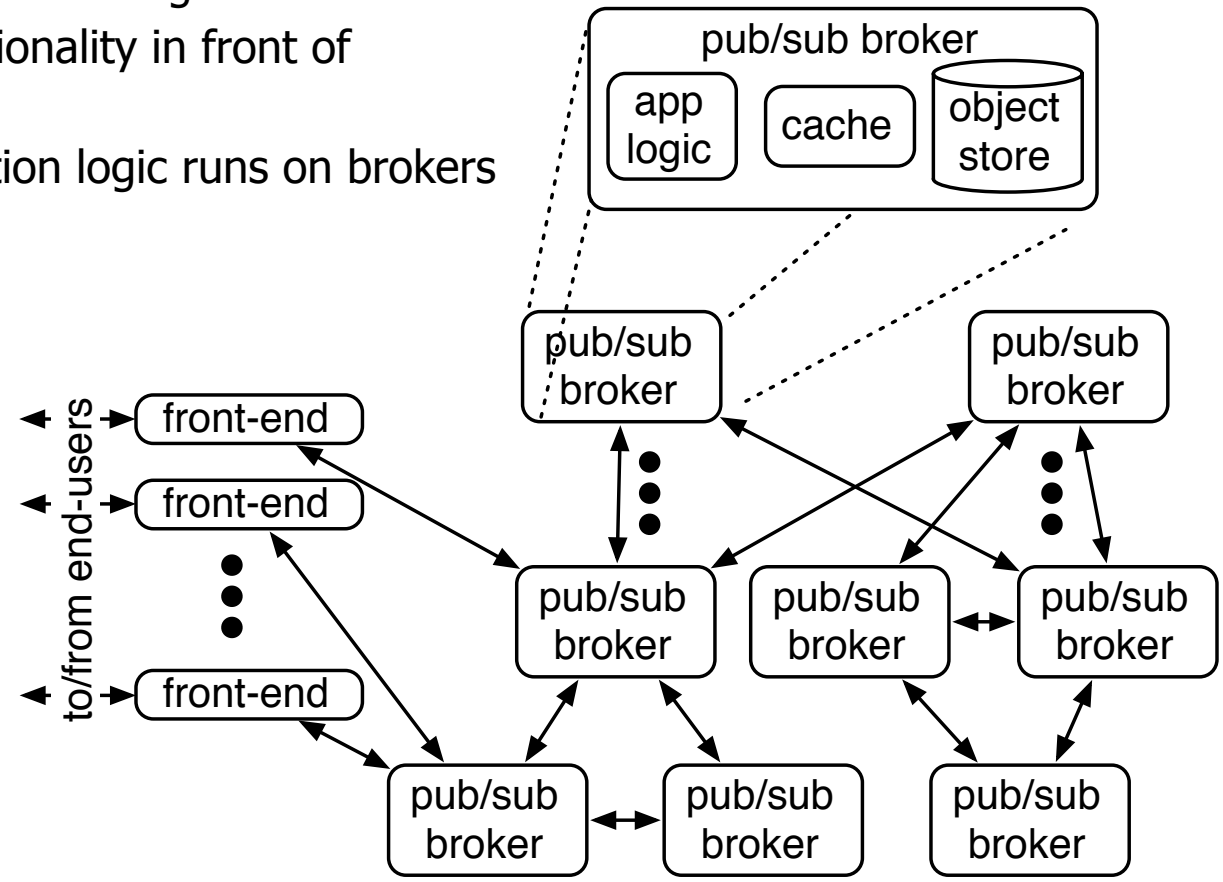


Publish/subscribe Architecture

Merge cache and storage layer of storage-centric architecture

Augment brokers with storage and application logic

- Distribute object store throughout brokers
- Include cache functionality in front of object store
- Ensure that application logic runs on brokers



Benefits of Pub/sub Architecture

Responsiveness

- Push-based architecture: brokers can respond to new data immediately
- Run application logic on broker nodes (unlike memcached)
  - e.g.: efficient dynamic computation: who is commenting on user's posts now

Scalability and elasticity

- Add more machines to broker network
  - Publish/subscribe broker network routes over all nodes
- Global scaling up only involves changing local data

Load balancing

- Platforms must adapt to changing patterns of end-user behaviour
  - Traffic spikes: flash crowds & content "going viral"
- Distributed publish/subscribe architectures inherently provide load-balancing
  - Multi-hop routing spreads load
  - Fine-grained, content-based classification of data spreads load

Support for Third-Party Real-Time Apps

Third-party apps are hosted at brokers

Sensible model for third-party applications:

① Application providers **retain ownership of data**: do not give it away
- Facebook currently do not run extensions on their servers

② Third-party applications **only see required data**
- Benefits privacy and facilitates payment plans based on actual usage
- Expressive subscription languages mean that third-party apps do not filter data

③ New applications **scale by adding message brokers**
- Preserves scalability and elasticity even as third-party applications join platform

Open Challenges

Architecture not storage-centric: complicates **persistence**

- Need to manage live and historic data uniformly
- Requires careful monitoring of replication across availability zones

New algorithms for **request routing** needed

- e.g. load-balancing of request flows in broker network
- Static vs dynamic decisions, maintenance of broker topology

Security harder to enforce

- Third-party code executes as part of core infrastructure
- Relies on sand-boxing for data and performance isolation

Conclusions

Abandon storage-centric view and embrace on-the-fly processing

Distributed pub/sub system as backbone for social web platforms

- Satisfies increasing demand for fresh data processing
- Supports on-the-fly data analysis by third-party applications

Support for scalability, elasticity, and load balancing

- Provides more uniform architecture for scaling
- Facilitates optimisation of data routing strategies

Thank You! Any Questions?

Peter Pietzuch
<prp@doc.ic.ac.uk>
http://lsds.doc.ic.ac.uk