LSDCS – Large-scale Distributed Computing Systems

Hosted by
CITI-Center for Informatics and Information Technologies of FCT/UNL
and
Departamento de Informática da Faculdade de Ciências e Tecnologia da UNL (DI – FCT/UNL)

Generic goal of the team

Research methods and tools to deal with computing systems interacting at the wide area level, namely, systems integrating a large number of stationary and mobile computers.
Specific goals of the team

- Large-Scale and Mobile Computing CSCW Software Systems
- Data management for Large-Scale and Mobile Computing Environments
- Overlay Networks and P2P systems
- Security Issues for Peer Group-oriented and P2P scenarios

Permanent staff

- http://asc.di.fct.unl.pt/lsdcs
- José Legatheaux Martins, PhD (jalm@di.fct.unl.pt)
- Henrique João Lopes Domingos, PhD (hjd@di.fct.unl.pt)
- Nuno Manuel Ribeiro Preguiça, PhD (nmp@di.fct.unl.pt)
- Sérgio Marco Duarte, Research Assistant (smd@di.fct.unl.pt)
Outline of the presentation

- Roots of the team
- Current projects
  - Databricks and ICE3 cooperation
  - Mobisnap
  - DEEDS
  - Mobiworkflow
- Future research directions

Roots of the team


- The objective of the DAgora project was to create a platform to support the development of collaborative applications for large-scale environments that included mobile computers

DAgora = Distributed Agora
DAgora project main goals

- CSCW for Large-scale and Mobile Computing
- Focus on Support for:
  - Large-Scale, High-Availability, Variable QoS
  - Reliable Communication and Distribution
- One Phd thesis, two MSc thesis, several papers, some prototypes, ...

From 2000 up to now

- The team has been engaged in several funded research projects
  - Databricks – Extending the data management part of DAgora (with the Programming Languages team of CITI)
  - Collaboration with the ICE3 project (Marc Shapiro - Microsoft Research, UK)
  - MobiSnap – Data management solutions for mobile databases (with Universidade do Minho, Portugal)
  - DEEDS – Active Overlay Network for Event-dissemination (only locally funded)
  - MobiWorkflow – CSCW Software System for the Mobile Computing Environment
Databricks (2000 – current)

This project focuses on the design and implementation of adequate system support to ease the implementation of new applications for mobile environments.

To this end, it provides, not only a set of data-management solutions suitable for different mobile settings, but also a specialized component-based framework and associated linguistic support that allows programmers to easily create adequate data-management solutions for their applications.

The main teams’ contribution - DOORS

- DOORS (DAgora Object Oriented Repository) is an evolution of the data management sub-system of the DAgora platform

- General approach
  - Data model: objects (named coobjects)
  - Architecture
    - Replicated servers
    - Epidemic propagation
    - Full clients
    - Partial caching
  - Optimistic replication: read any/write any
  - Log propagation
    - Update: method invocation
DOORS Architecture

DOORS framework

- Motivation
  - Different applications require different solutions
    - Reconciliation
    - Handling awareness information
  - Good solutions can be reused in similar applications
- Approach
  - DOORS component framework decomposes object operation in several components
    - Programmers can select which data management techniques to use
Blind invocation

- Motivation
  - *Cache misses* usually prevent any data access
  - Users can *still* execute useful work
- Approach
  - Users can submit operations on objects (subobjects) that are not locally cached (if they have a reference)
  - Replacement object can be used to observe the result of operations
Blind invocation

Integration of synchronous and asynchronous activities

- Motivation
  - Data must be handled in asynchronous and synchronous sessions
  - Intrinsic differences between the two settings
    - Coordination and awareness information
- Approach
  - Allow co-objects to be shared synchronously (besides being modified asynchronously)
  - Allow different techniques to be used in different settings
    - Different operations (granularity)
    - Different techniques: reconciliation, handling awareness information
Integration of synchronous and asynchronous activities

Group communication

DOORS Coobjects are build as a ComponentJ Component

component SchedulerCoobject {
  provides ICapsule capsule;
  intro T2vCapsule cap = Cluster2vCapsule;
  intro TLogCore logCore = LogCoreSeq;
  intro TAttrib = SeqAttribGolding;
  intro TAwareness = Awareness;
  intro TCluster clustCommit = ClusterSimple;
  intro TCluster clustTentative = ClusterSimple;
  ...
  plug cap.logCore into logCore.logCore;
  plug cap.clusterCommit into clustCommit.clustCommit;
  ...
  declare m { // this method populates the cluster with data objects
    void init(ICluster cluster) {
      cluster.add(new SchedulerData);
    }
  }
  plug m into clustCommit.datainit;
  ...
}
In ComponentJ components are recursively composed of components.

SchedulerCoObject

Cluster
- cluster: ICluster
- data: IData
  - SchedulerData
  - LogCoreSeq
    - attr: IAttr
      - SeqAttribGolding
        - attr: IMulti
          - LogCore:
            - logCore: ILogCore
              - logOrdCommit: ILogOrder
              - logOrdTent: ILogOrder
              - attrib: IAttrib
              - capsule: ICapsule
                - Cluster2vCapsule
                  - capsule: ICapsule
                    - clusterCommit: ICluster
                    - clusterTentative: ICluster
                    - aware: IAwareness
                    - logCore: ILogCore
                    - logOrdCommit: ILogOrder
                    - logOrdTent: ILogOrder
                    - attrib: IMulti
                    - capsule: ICapsule
                      - Cluster:
                        - data: IData
                          - Cluster
                        - cluster:
                          - data: IData
                            - Cluster

Runtime Architecture

Generic CoObject with Factory

DAgora System
- ICapsule capsule;
  - MyCoobjectFactory fact;
  - capsule = fact.create();
- ...
  - Cluster c = capsule.getCluster();
    - DataObj d = c.getObject(ObjID);
    - ...

MyCoobjectFactory (Java)
- create() {
  - SchedulerCoobject.createInstance();
  - return SchedulerCoobject.cap;
- ...
  - _Implementation (Java)
    - Implements I...
    - Capsule Implementation (Java)
    - Implements ICapsule
    - Implements ICluster
    - Implements ..._Implementation (Java)

MyCoobject (ComponentJ)
- cap : ICapsule
  - Components
  - Cluster Component
    - Cluster Implementation (Java)
  - Capsule Component
    - DataObject_Impl
    - DataObject_Impl
Mobisnap – Mobile Snapshot Management

The main goal of the Mobisnap project is to provide methodologies and tools to deploy relational database applications on mobile computers.

The project’s main contributions are:

1. Mobile transactions - the main mobile update mechanism
2. Reservations - the mechanism used to provide guaranties about the ability of the mobile transaction to commit when finally executed by the central server, as well as a way of preventing replica divergence.

Mobisnap approach

- General approach
  - Data model: relational data model
  - Architecture
    - Single server
      - Official database state
    - Multiple full clients
      - Partial caching
  - Optimistic replication: read any/write any (client)
  - Updates
    - PL/SQL programs (dubbed: mobile transactions)
    - Include support to notify users
  - Evolutionary middleware approach
    - Legacy client may access the database directly
Mobisnap system model

- Mobisnap server
  - mobile trx. interpreter
  - reservation subsystem
  - comm. subsystem

- RDBMS
  - primary
  - replica

- legacy client

- Mobisnap client
  - mobile trx. interpreter
  - pre-fetch subsystem
  - comm. subsystem

- DB replica

- Mobile transactions

  - Updates expressed as small PL/SQL programs
    - Subset of PL/SQL with minor changes

  DECLARE
  l_stock INTEGER;
  l_price FLOAT;
  BEGIN
  SELECT stock,price INTO l_stock,l_price FROM products WHERE id = 80;
  IF l_price <= 100 AND l_stock >= 10 THEN
    UPDATE products SET stock = stock - 10 WHERE id = 80;
    INSERT INTO orders VALUES (newid,8785,80,10,l_price,'processing');
    NOTIFY( 'SMS', '351927435456','order added for product red...');
    COMMIT (80,l_price);
  ENDIF;
  NOTIFY( 'SMS', '351927435456','Order failed...');
  ROLLBACK;
  END;
Reservations

Motivation
- Final results determined after reconciliation
- Important to obtain results independently

Approach
- Reservations provide promises about the future database state
  - Different types of reservations
- Clients obtain reservation (before disconnecting)
- Clients automatically verify if the result of mobile transactions can be guaranteed using available reservations
- Servers guarantee that reservations’ promises are not broken

Reservations

BEGIN
SELECT stock, price INTO l_stock, l_price FROM products WHERE id = 80;
IF l_price <= 100 AND l_stock >= 10 THEN
  UPDATE products SET stock = stock - 10 WHERE id = 80;
  INSERT INTO orders VALUES (newid, 8785, 80, 10, l_price, 'processing');
  COMMIT (80, l_price);
ENDIF;
ROLLBACK;
END;
**SqlIceCube**

- **Motivation**
  - Semantics-based reconciliation leads to better reconciliation results
  - Programmers must expose semantics

- **Approach**
  - Reconciliation as an optimization problem (extends IceCube)
    - Heuristic search directed by semantic relations
  - Semantic relations inferred automatically from the code of mobile transactions
    - Static analysis of transaction text
      - Per-transaction: extract read sets, write sets, preconditions
      - Between pairs of transactions: infer relations

---

--- **INSERT ORDER** -------------------------------

```
SELECT stock, price INTO l_stock, l_price FROM products WHERE id = 80
IF l_price <= 100 AND l_stock >= 10 THEN
    UPDATE products SET stock = l_stock - 10 WHERE id = 80;
    INSERT INTO orders VALUES (newid, 80, 10, l_price, 'pending');
    COMMIT;
ELSE
    ROLLBACK;
ENDIF;
```

--- **CANCEL ORDER** -------------------------------

```
SELECT status INTO l_status FROM orders WHERE id = 3;
IF l_status = 'pending' THEN
    UPDATE orders SET status = 'cancelled' WHERE id = 3;
    UPDATE products SET stock = stock + 3 WHERE id = 80;
    COMMIT;
ELSE
    ROLLBACK;
ENDIF;
```

---

*Do not commute*

**Cancel** writes and **Insert** reads

(products, stock, id=80)
### INSERT ORDER

```sql
SELECT stock, price INTO l_stock, l_price FROM products WHERE id = 80;
IF l_price <= 100 AND l_stock >= 10 THEN
    UPDATE products SET stock = l_stock - 10 WHERE id = 80;
    INSERT INTO orders VALUES (newid, 80, 10, l_price, 'pending');
    COMMIT;
ELSE
    ROLLBACK;
ENDIF;
```

### CANCEL ORDER

```sql
SELECT status INTO l_status FROM orders WHERE id = 3;
IF l_status = 'pending' THEN
    UPDATE orders SET status = 'cancelled' WHERE id = 3;
    UPDATE products SET stock = stock + 3 WHERE id = 80;
    COMMIT;
ELSE
    ROLLBACK;
ENDIF;
```

---

**DEEDS**

DEEDS is born from the requirements of the DAgora platform for event dissemination. This kind of service is crucial for the support of large scale collaboration.

DEEDS is an event dissemination platform allowing the multicasting of events, in a large-scale scenario, with different quality of service requirements.

DEEDS is an active overlay network based on a backbone of (several hundred) servers and many (thousands of) clients.
DEEDS - In a nutshell

An extensible event and data dissemination platform based on active overlay networking principles.

DEEDS - Main goals

- Extensibility
  - quality of service
- Large-scale support
  - WAN & number of users/nodes
- Heterogeneity support
  - overlay network infrastructure
  - mixed protocols (tcp, udp, multicast, http, etc)
DEEDS - Dissemination model

- Paradigm *publisher/subscriber + feedback*
- Generic events
  - payload (private/opaque)
  - envelope (meta-data)
  - receipt (system-level data)
- Generic Filtering
  - programmable on envelope contents
- (Active) Event Channels
  - multiple independent flows
  - specific quality of service

DEEDS - Active Event Channels

- Logical address
  - event flow structure
  - rendezvous point
- Broker
  - fully decoupled interactions between parties
- Specific QoS attribute
  - reliability, ordering, persistency, addressing, etc.
  - underlying protocol independency
- Active networking module/template
DEEDS - Dissemination Infrastructure

- Active overlay network
  - 3 tiers/contexts
- 3 node types
  - backbone server nodes
  - secondary server nodes
  - client nodes
- Heterogenous links
  - "transports"
    - tcp, udp, http, email, etc.

DEEDS - System Registry

- Local data repository
  - Persistent (configurations)
  - Volatile (soft-state/containers)
- Global data repository
  - Event channel directory
  - Event channel QoS templates
    - Accessible on every node
    - Cached on client nodes
      - on demand (lookup side-effect)
    - Replicated (persistently) on server nodes
      - on demand (lookup side-effect)
      - proactively (low priority, low bandwidth trickle process).
DEEDS - Node Architecture

- Execution Engine
  - Event channel instances
    - routing, filtering, soft-state housekeeping (QoS chores)
  - Transport instances
    - low-level packet exchange
  - System-level services
    - Network monitoring (backbone tier)
    - System registry related tasks (updating, lookup)
- Containers
  - Local soft-state monitoring and notification (routing tables, subscribers lists, etc)

DEEDS - Backbone monitoring

- Assemble and share a "consistent" global view of the 1st tier
  - number and identification of nodes
  - relative distances among nodes
  - allowed transports
- Hello service
  - Latency measurements, node failure detection
- LinkState service
  - Efficient dissemination of local Hello data.
DEEDS - Backbone monitoring
DEEDS – Latest developments

- Extensibility Validation

- Event channel QoS module design
  - unfiltered core-base multicasting
    - single/multi source, reliable/unreliable, total ordered
  - generic, filtering-aware multicasting
    - source-based, core-based

DEEDS – Future developments

- Continued Extensibility validation
  - Large-scale P2P unicast and multicast algorithms
  - CAN and content routing algorithms

- More case-study application scenarios

- Polish the prototype implementation
MobiworkFlow Project

Goal: Support (architecture, mechanisms and coordination facilities) for collaborative-oriented dynamic and mobile workflow management systems

Dynamic:
the workflow (business/organizational) process coordination and definition is refined, while the process is evolving

Mobile:
tasks are performed by a group of users distributed in a stationary as well as in a mobile internetworked environment
WFMS: process based on a composition of task-objects
Task-objects: replicated by groups of users

Collaborative-oriented:
users performing tasks can work collaboratively
(resources-data sharing, awareness control/notification mechanisms in a peer-group basis)
**MobiworkFlow: main contributions**

1. Architectural model and support components  
   Groups of users/devices in an hybrid environment  
   (stationary vs. mobile network environment)

2. Coordination model for dynamic (adaptive) workflows  
   in a mobile environment  
   • WPDL / coordination model and tools  
   • Workflow Process: composition model to express  
     sequences (with pre-cond/post-cond and  
     constraints) of task-services  
   • Fault/Exception Handling and compensation  
     mechanisms

3. Integration model for mobile (web-service-enabled)  
   transactions  
   • Web-enabled (ext. WSDL-ext support)  
   • Mobile transactions - mobisnap style (?)

---

**MobiWorkFlow Model**

Mobile workflow process:  
A non-repetitive (or acyclic) activity composed by a  
sequence of tasks, performed by a group of users  
involved in the implementation of a common goal.

Tasks can re-instantiate new workflows (in a recursive  
model). Final tasks can be performed by mobile users  
in a collaborative-way
MobiworkFlow: coordination

The workflow process is a coordination context, shared and extensible, propagated (all or part of) to the distributed participants. Final tasks to be performed by the users are work units, fetched and locally replicated. Locally, the mobile participants interpret the coordination context, fetch, enact and perform the assigned tasks. Tasks can be self-contained (data, forms) or can bind to specific (external) applications to be locally executed.
**MobiworkFlow: coordination**

Coordination involves:

The coordination model and a WPDL (inspired in PDPL1.1 by WfMC), to express:

- Semantic information (pre-conditions, post-conditions and constraints, anticipating problems) to regulate concurrency control mechanisms, task-objects replication, reconciliation of results and conflict detection/resolution
- A registration service used by the participants to register resources and information used by coordination protocols
- A notification service (event-channels for group-awareness control)
- Dynamic RBAC model

**MobiworkFlow: transactions**

- Concurrent Mobile transactions (task-object updates)
  - Mobisnap style (?)
- Relaxed ACID semantics / optimistic propagation combined with the coordination context
- SOAP/HTTP(S) supported
  - WS-oriented transactions service support
  - Leverages WS-Coordination support by defining two specific transaction types:
    - Short running atomic transactions (all-or-nothing) to propagate work-unities (task-objects)
    - Long-lived transactions associated to changes in the coordination process
Future directions of research

Ongoing work:

- Mobile workflow management
  - The MobiWorkFlow project is still ongoing
- Data management for large-scale systems
  - Some extensions of the work in DOORS and in ICE3 are planned
  - Finalize the integration with ComponentJ
  - New caching methods for web-based applications
- Active overlay network for event dissemination
  - The work on DEEDS is still ongoing

Future plans:

- Very large-scale systems
  - Data management solutions for very large scale systems – updateable P2P DHTs
  - Hybrid solutions for routing in P2P systems
  - Data dissemination, collaboration in very large systems
  - Support for secure peer-group and communication
  - Security based on distributed trust/reputation models

- Improve international cooperation
- New funded project will be proposed