RPC-V: Toward Fault-Tolerant RPC for Internet Connected Desktop Grids for Volatile Nodes

Samir Djilali, Thomas Herault, Oleg Lodygensky, Tanguy Morlier, Gilles Fedak and Franck Cappello
Fault Tolerant RPC

- RPC is one of the programming models envisionned for the GRID

- In Internet-Connected Large Scale Grids, failures are not rare events
The Challenges of Large Scale

- Volatility
- No Stable Components
- Intermittent Crashes
- Asynchronous (best effort) networks like Internet
- Connectionless Interactions
- Changes of the System Size

Crashes + Asynchrony

Impossibility of Consen
Consequences

- Theoretically impossible to do consensus in a deterministic way
- Consensus is simpler than Statefull RPC
  - By defining the order of RPCs for one server could solve a consensus problem
- Either circumvent the problem
- Or try to address another problem
  - Stateless RPC
  - (or per-client-basis statefull RPC)
Related Works

- **Fault Tolerance for the Grid**
  - 3 tier FT Arch.
  - GridRPC
  - Ninf, RCS, NetSolve

- **Fault Tolerance & Large Scale**
  - Probabilistic Solutions

- **Fault Tolerance & Remote Procedure Calls**
  - Mostly using reliable or LAN components
RPC-V FT-Protocol

- **Preventive Actions**
  - Each component locally logs every communication
  - Synchronizes and/or replay lost events on communications
  - Coordinators use passive replications

- **On Suspicion**
  - User may re-launch clients when they suspects clients to be unreachable
  - When server or client suspects coordinators they change of preferred one
  - When a coordinator suspects server failure, it schedules RPCs again
  - When a coordinator suspects another coordinator, it recomputes the topology
Implementation

- **Over XtremWeb**
  - Heart-Beat
  - Includes the three tiers: client, coordinator (dispatcher), server
  - Connection-less communication protocol

- **Coordinators Topology**
  - Ring (initially, faults may change the topology)
  - Initial list, updated using state synchronisation and/or user actions

- **State Synchronisation**
  - Between Client and Coordinator, or Server and Coordinator: maximum timestamps
  - Between Coordinators: peer-wise timestamps comparisons
Implementation

- **Message Logging**
  - Pessimistic/Optimistic message logging (sender-based)
  - Garbage Collector, based on long delays or on user actions

- **Coordinator Scheduling**
  - First Come, First Served
  - Dates are shared to avoid unnecessary re-submissions from replicas
Performance Evaluation

- Experiments on Local Area Networks for parameter isolation
  - Reproducibility of experiments
  - To highlight System overhead and FT capabilities
- Real Life Experiments
  - Validity of the implementation
  - Scalability test
LAN Experiments

- Cluster, 16 Servers, 4 Coordinators, 1 Client
- Ethernet 100 Switch, homogeneous Athlon XP 1800+, 1Gb RAM, IDE Disks
- Synthetic Benchmark
  - non blocking RPC
  - execution time (low to stress the comm.)
  - parameter size
  - result size
- Fault Generator
  - Upon order, or regularly.
  - 5s heartbeat / 6 miss => failure
Comparison of the three message logging strategies

![Graphs showing comparison of message logging strategies.](image)
Coordinator Replication Time

![Graph 1: Time vs. Data Size (Bytes)](image1)

![Graph 2: Time vs. Number of Calls](image2)
Synchronization Time

- Blue cross: using client logs only
- Red cross: using coordinator logs only
Fault Frequency Impact

![Graph showing time (sec.) vs. nb faults/minute for faulty servers and faulty coordinators.](image)
Real Life Experiments

- **International Wide Experiment**
  - Polytechnic School of Lille (France)
  - University of Wisconsin (USA)
  - Paris-Sud University (France)
- ~120 Servers (160 CPUs)
- “Real life” production application of Alcatel (validation and evaluation of communication networks)
Distribution of Tasks Duration

![Distribution of Tasks Duration](image-url)
Coordinator Replication Time

![Graphs showing the replication time in Real Life and Confined environments.](image)

- **Real Life**
- **Confined**
Reference Execution without Faults
Execution with two consecutive coordinator faults
Conclusions & Futur works

- Reserve most stable resources to computing nodes (servers) and not to architecture (coordinators)
  - Due to time gap between server execution and traversal of the system by a RPC
- Larger tests, optimization for larger scale
- Impact of checkpointing
Conclusion & Futur Works (with respect to asynchrony)

- **Conservative assumption**: Stateless RPC
  - “per-client-basis” statefull is OK
- **Evaluate on GRIDs the asynchrony**
  - Can we conceive better FD than TCP connection?
- **Use gossiping / probabilistic techniques to overcome the limitation**