Distributed Systems

Olivier Gruber

Full-time Professor
Université Joseph Fourier

Senior Researcher
Projet SARDES (INRIA et IMAG-LSR)

Who am I?

- INRIA – Rocquencourt, France
  - Ph.D. (1992)
  - INRIA Team Leader (1992-1995)
- IBM Almaden Research Center (1995-1996) – California, USA
  - Full-time researcher - advanced database group
- IBM Watson Research Center (1997-2007) – New York, USA
  - Object-oriented and component-oriented systems
  - Web middleware and pervasive ecosystems
  - Senior researcher and technical advisor to IBM strategists
- Full-time Professor (2007-today) – Grenoble, France
  - Joseph Fourier University, Grenoble
  - SARDES team, INRIA Rhones-Alpes

Acknowledgments

- Prof. Sacha Krakowiak
  - Used his lectures as a canvas
- Reference Book

  Distributed Systems
  Principles and Paradigms

  Second Edition

  Andrew Tanenbaum and Maarten Van Steen

- Research Articles
  - Cited on various slides

This Year Outline

- Course Goals
  - Understand architecture and design trade-offs
  - Master core techniques and essential distributed algorithms
  - Discuss existing systems and frameworks
- Today
  - Background on distributed systems
  - Fundamentals – Part One

Distributed Systems

- What are they?
  - Collection of cooperative entities
- Humorous Definition from L. Lamport

> ***A distributed system is one that stops you from getting any work done when a machine you've never heard of crashes.***
> ***Leslie Lamport***

  - Highlights the cooperative nature of distributed systems
  - The increased probability of failures
  - The likeliness of their consequences on overall availability and human experience

Failure Examples

- Buffer Change at Polygram
  - A small buffer size change, failure of the order-shipping workflow
  - Hundreds of trucks and employees out of work for more 24h
  - Intrinsic costs and warranty violation
- September 11th, 2001
  - Most businesses in the towers had only regular data backups
  - No disaster recovery from replicated data
- Space Shuttle
  - Four computers, many missions ended with one left working...
- Ariane 501
  - June 4th 1996, first launch of Ariane 5 fails: Ariane 5 explodes

  http://www.cnes.fr/espace_pro/communiques/cp96/rapport_501/rapport_501_2.html

Distributed Systems

- Examples
  - Networked workstations
    - A typical Local Area Network with distributed applications
    - Distributing processing or sharing data
  - The World-Wide Web
    - Where world-wide scalability is *the* challenge
    - Client-server or peer-to-peer
  - Cellular wireless networks (telephony)
    - For voice and data, mobile devices
    - Health monitoring of patients at home or travelling

Distributed Systems

- More Examples
  - Game Consoles
    - Sony PlayStation 3 was originally designed to deliver 1 teraflops
    - Four processors, highly-parallel flow-oriented machine
  - Embedded networks
    - In planes or cars
      - BMW Serie 7
      - 4 networks, 70 computers
      - 70% of car failures are computer-related (hardware and software)
    - Sensor networks
  - On-chip networks
    - Distributed systems on chip
    - Soon, more than 64 nodes interconnected on one silicium chip

Distributed System Origins

- Hardware Revolution
  - Processing
    - From 10 millions of dollars, 1 instruction per second
    - To a few hundreds of dollars, 1 billion intructions per second
    - Rolls Royce:
      - Would be a dollar
      - Would get a billion miles per gallon
      - Would be the size of a match box
  - Networking
    - From some 300 bps (early modems)
    - Ethernet from 10Mbps to 10Gbps, wireless 54 Mbps or more
    - Latency from a few microseconds to a few hundred milliseconds
  - Storage
    - Access time around a few milliseconds (5 to 10ms)
    - External transfer rate around 300Mbps

Distributed System Origins

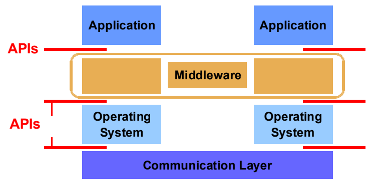
- Software Revolution
  - From standalone applications to cooperative applications
- Standalone Application
  - Sweet spot for traditional operating systems
  - Its own data, its own processing, its own windows
- Cooperative Applications
  - Integration and interoperability
  - Share data (like a shared file system or database system)
  - Exchange messages like email systems, SMS, web browsers or X11
  - Cooperate like systems embedded in a car or world-wide banking systems

Distributed System Challenges

- Software
  - ***Software is lagging behind hardware, incredibly so!***
    - Distributed programming is orders of magnitude harder
  - Reasons:
    - Parallelism, asynchronous, communication latency, failures, etc.
  - Should impacts
    - Programming languages and models
    - Tools and runtimes
    - Algorithms
  - Usually
    - Approached through a middleware...

Distributed System Challenges

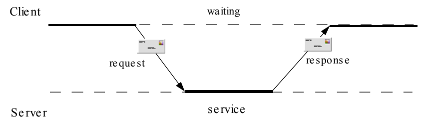
- Introducing Middleware
  - Higher-level APIs, attempts to help...
  - All differents... all quite complex...
- Essentially Two Middleware Families
  - Message-oriented
  - Object-oriented



Distributed Programming

- Message-Oriented Paradigm
  - Send and receive messages
    - A message is a byte stream of known length
    - Sender: build and send a message
    - Receiver: wait and receive a message
  - Both synchronous or asynchronous
    - Sender may not wait for the response
    - Receiver may not wait for the receive

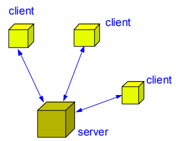
Distributed Programming



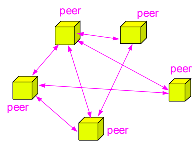
- Client-Server Basics
  - Mostly a synchronous world
    - Make a request to a server, wait for the response
    - But not always, like backgrounded image downloads for web pages
  - Relies on
    - Naming scheme: names the destination of messages
    - Routing scheme: routes messages to their destination

Distributed Programming

- Client-Server Architecture
  - Simpler approach to distributed programming
    - The Web is a perfect example
    - Each client independently interacts with servers
  - Not fully distributed
    - Each server essentially provides a centralized decision point, but also a single point of failure



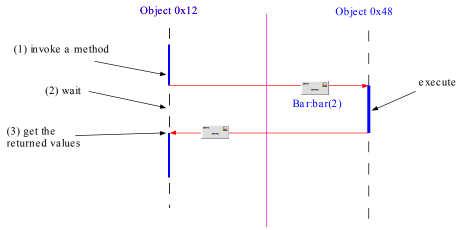
- Distributed Peer-to-Peer Architecture
  - Towards identical processes
    - No more clients or servers
    - Only identical peers
  - Engaged in a cooperative process
    - Exchange data
    - Execute logic



Distributed Programming

- Object-oriented Paradigm
  - Remote objects and method invocations
    - Traditionally synchronous, could be asynchronous
    - Message is the method invocation (argument marshalling)
    - Routing is based on object identity
  - Built on a message layer
    - Provides better language integration
    - Claims improved developers' productivity
    - Java RMI or Jini are examples

Discussing Programming Models

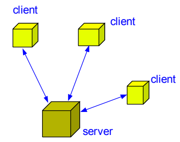


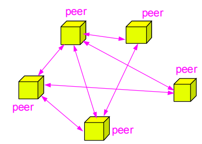

```
public class Foo {
  Bar m_bar;
  public int foo() {
    return m_bar.bar(2);
  }
}
```

```
public class Bar {
  int m_value;
  public int bar(int factor) {
    return factor*m_value;
  }
}
```

Distributed Programming

- Client-Server Architecture
  - Objects can act as servers
    - May be grouped in physical servers
    - A Web server could be implemented as a remote Java object, accessed via RMI
- Distributed Peer-to-Peer Architecture
  - Objects can be seen as peers
  - Objects may implemented distributed protocols
  - Objects may be replicated and cached





Distributed Programming

Distributed Programming...

Why is it so hard?

Traditional Programming

- Time
  - There is a notion of time: the hardware clock
  - This means that all events happen on one timeline
- Memory
  - Reads and writes are consistent
  - Assumed to be fast
- Processing
  - Method invocations or function calls
    - Synchronous and expected to work (no remote failure)
  - References are expected to stay available
    - No loss of in-memory data structures
  - Often single threaded logic

Distributed System Challenges

- No Global Time
  - Only causality applies
  - Calls for asynchronous models
- No Global Ordering
  - Between senders and even between messages
  - A simple loop with a method call suddenly does not work as expected anymore...
- No Global Consistency
  - In practice, too costly and difficult
  - The Web caching example

Distributed System Challenges

- Failures
  - Lost messages or method invocations
    - Distinguish long delays from actual message loss?
    - Distinguish message loss from actual node or process failure?
  - Lost remote references
    - Violates GC assumption
  - Consistency
    - Difficult to achieve synchronization on shared objects/ data
    - Propagating updates between copies of shared data
- Security
  - Becomes rapidly a concern
    - Eavesdropping on communication
    - Identity theft
    - Trusting the middle man...

Overall Goals of Distributed Systems

- Transparency
  - Access, location, migration and relocation transparencies
  - Concurrency and fault-tolerance
- Scalability
  - Geographical scale
  - Scaling in size (users, nodes, resources)
  - Administrative scalability across administration domains
- Availability
  - Facing failures or downtime
  - Facing evolution as long-live systems must change
- Mobility
  - Users are mobile, across the globe, with intermittent connectivity

Essential Trade-Offs

- About Transparency
  - Transparency is considered
    - To be more productive
    - But usually expensive to provide
  - Not always better
    - Can't change the laws of physics
      - A few hundred milliseconds across the atlantic
      - Different time zones, time and geographically sensitive services
    - Knowing what is costly
      - Actual costs do impact algorithms and data structures
    - Knowing where failures may happen
      - Leverage application semantics
      - Levarage the ability of humans to adapt

Essential Trade-offs

- About Scalability
  - Scale in number of nodes or users
    - From a few nodes to thousands of nodes...
    - The Web... millions of nodes... such as Gnutella with 50 millions peers
  - Scale geographically
    - Physical network capabilities are a concern
      - *The speed of light can't be changed...*
      - Limited bandwidth and latency
        - Latency is more of a problem than bandwidth for distributed systems
        - Worsen by the fact that most distributed systems are synchronous
    - Communication on WANs
      - Unreliable: loss of messages, partitioning, non-FIFO channels
      - Point-to-point channels (no multicast or broadcast)
  - Scale administratively
    - Conflicting policies for resource management and payment
      - Different requirements about security
    - Trust between administration domains

Essential Trade-offs

- About Failures
  - Automated fault-tolerance is more productive
    - 80% of the code of a DBMS is impacted by transactions and recovery
    - Error-prone issues for most developpers
  - But fault-tolerance is expensive
    - Synchronization in distributed system are complex algorithms
      - A lot of messages are exchanged
      - Supporting message loss incurs extra complexity
    - Recovery means logging on stable storage
      - Still expensive, even with faster hardware

Conclusion

- Client-Server Architecture
  - Incredibly successful in the last 10 years or so
    - Supports the Web and its related e-commerce activities
    - Both Business-to-Consumers (B2C) and Business-to-Business (B2B)
  - It can scale well and provide high-availability
    - It is a matter of technology
      - Fast hardware improvements, smart in-network caching, and router technologies
    - It is therefore a matter of money
      - But also a matter of design for both the middleware and its applications

Conclusion

- Beyond Client-Server Architecture
  - Why?
    - Not all communities have enough money
    - Not all systems can accept single points of failures
      - Each web server is a failure point, unless it is replicated (which we will study)
    - Not all systems can work across uncooperative servers
      - Global decisions and cooperations are often unavoidable
      - Examples: banking, financial systems, trading, booking systems
  - Towards fully distributed systems
    - Fundamentally a peer-to-peer architecture
      - Essentially about looking at equal partners in a distributed system
      - This is not only about file sharing, it is about more advanced algorithms
    - Addressing exciting transparency and correctness challenges

Conclusion

- Course Content Overview
  - Fundamentals
    - Message-oriented paradigm
      - Naming destinations and routing messages
      - Discussing and mastering time
      - Synchronization, including election algorithms
      - Memory consistency models
      - High-availability and fault-tolerance through replication
    - Object-oriented paradigm
      - Object-oriented paradigm, type reflection, class loaders
      - Object identity, object proxies, parameter marshalling, distributed garbage collection
      - Service-oriented architecture and modules for networked managed platforms
  - Case study:
    - Java Messaging Service
    - Java Platform, RMI, and OSGi