Lecture 23: April 23
CMPSCI 677 Operating Systems - Spring 2018
Lecturer: Prashant Shenoy

Scribes:
Aaron Elliot
Krishna Prasad Sankaranarayanan

May 1, 2019

Subjects Covered

- Distributed Objects
- EJB's
- CORBA
- .NET
- Jini
- Distributed Data Processing

1 Distributed Middleware

1.1 Distributed Objects

In case of remote objects, code on a client machine wants to invoke an object's method on a server machine. A common way to achieve this, is as follows. Clients have a stub called proxy with an interface matching the remote object. An invocation of a proxy's method is passed across the network to the 'skeleton' on the server. That skeleton invokes the method on the remote object and returns the marshalled response. This can be recognized from earlier in the course as an RMI or RPC call.

Distributed objects are similar, but the distributed objects are themselves partitioned or replicated across different machines. Distributed objects use RPC. Middleware systems have been developed to support distributed objects. However, middleware systems were developed originally for complex distributed applications but nowadays, they are only utilized in a client-server framework. Since they were developed for complex applications they are often heavy weight, and some became commercial failures; e.g. CORBA.

1.2 EJB - Enterprise Java Beans

It is basically standard Java added with services like RMI, JNDI, JDBC(used to connect to Databases), JMS(Java Messaging Service)

Middleware services also provide entity beans which are essentially persistent objects. The lifetime of a persistent object is independent of the lifetime of the process. The state of the object is stored on the disk which can be reconstructed when necessary.

EJB are fundamentally object oriented, with two components, the interface and the implementation.

1.2.1 Types of EJB's

1. Stateless Session beans - Object stays alive for the duration of the session. It has only methods.

2. Stateful Session Beans - Closest to Java objects with state and private variables.

3. Entity beans - persistent objects.

4. Message -driven beans - used to send/receive messages between objects.

1.3 CORBA - Common Object Request Broker Architecture

A the heart of CORBA is the Object Request Broker (ORB) [also called message bus] is a intermediate communication channel that allows communication between objects.

The intention for CORBA was to create the last middleware you'll ever need. They provide a dozen different services from concurrency to licensing in complex distributed systems. The advantage is that they can help reduce the code needed to develop complex distributed systems. However, in trying to provide every service you'd ever need, CORBA became very heavy weight. By becoming very heavy-weight, it became very difficult to learn and simple distributed applications would require deploying such a heavy weight system.

CORBA made many choices available to the developer. Interface Definition Language (IDL) Proxy is used to specify the objects and services. Object adapter provides portability between languages. Thus CORBA, is language independent. CORBA RPC's can be invoked as any type of Synchronous, One-way, or deferred synchronous. CORBA had a lot of flexibility, but that also came with complexity.

CORBA was not widely used. Linex's gnome desktop manager was made using CORBA distributed applications. CORBA was one of the first distributed middleware systems. Modern middleware systems take many ideas from it.

1.3.1 Event and Notification Service

CORBA included the publisher subscriber framework. Where suppliers/publishers post events to the event channel, and consumers/subscribers ask for events that they subscribe to from the event channel. Publisher subscriber works with any combination of Push and Pull. In CORBA it is a push -push model where data is pushed from Supplier to Event channel and subsequently to Consumer. In pull-pull mode, event channel polls data from the Supplier and similarly, Consumer pulls data from the event channel. To reduce polling, Asynchronous method invocation is used wherein a notification after data is arrived would be given. With respect to pure message overhead : Pure Push $<$ Asynchronous Pull $<$ Pure Pull.

1.4 DCOM - Distributed Component Object Model

COM is Microsoft's middleware which has now evolved into .NET [COM + OLE + ActiveX]. COM is a simple RMI based framework running local to a machine. It is mainly used for communication between Microsoft applications. Object Linking and Embedding (OLE) was added to allow microsoft office applications to communicate with one another via embedding and document linking. The ActiveX layer facilitates exposing these services as web applications. .Net has a language independent runtime, but ActiveX only works with Internet Explorer.

The archetecture of COM is fundmanetally the same as the distributed objects we describe above in section 1.1

1.4.1 Type Library and Registry

By default objects in DCOM are transient, but developer has option to make objects persistent on disks. Persistent objects are called Monikers.

1.5 Distributed Coordination System / Blackboard Architecture

Distributed applications can be classified based on whats happening in time and space dimension. Applications can either be coupled or decoupled in space and time.

1. $\langle$ coupled in space and time $\rangle$ Direct

2. $\langle$ coupled in space but not time $\rangle$ Mailbox - receiver is known but receiver state does not matter.

3. ⟨ coupled in time but not space ⟩ Meeting Oriented - unknown who will show up to meeting.

4. ⟨ decoupled in space and time ⟩ Generative Communication - components can communicate with another without knowing who might read it or when it would be read. Hence, loosest form of communication.

Publisher Subscriber/Bulletin Board architecture is an example of generative communication. A real world example of bulletin board architecture is a worker-pool popping jobs posted by clients to a shared job-queue.

1.5.1 Jini

It facilitates service discovery. These are also 0-Configuration Services. eg: Wifi. It uses a blackboard or bulletin architecture. Services advertise on the bulletin board and machines can access the services it requires through the board. In Jini, bulletin board is called JavaSpace or tuple space. Each tuple is a Java object. Essentially, Jini utilizes a Pub-Sub architecture with both Pull based as well as notification based discovery.

JavaSpaces can either be fully replicated or distributed. In case of replicated JavaSpaces , Writes need to be broadcasted to all replicas whereas reads are local. On the other hand, in distributed bulletin board, each board has a subset of nodes, while writes are local and reads need to be done on each and every board.

1.6 Big Data Applications

This is mostly covered in CS532 systems for data science. So we will only partially cover this.

Very large data sets are too large to process on one machine. We speed up processing by parallelizing it. MapReduce was the first big data processing framework. Map-reduce frame work has two phases, Map phase and reduce phase. In each phase a UDF is applied in a massive parallel step where many workers apply that UDF in parallel without communication. Many systems have followed, applying UDFs in massive parallel computation.

Apache Tez extended MapReduce to a directed acyclic graph of computations with recovery. Spark extended this to using Resilient distributed data sets partially in memory. Microsoft's Dryad and Naiad, Apache Kafka and Spark all added support for streaming data. Flink extended DAG model to allow for cyclic computations until all workers halt.

1.6.1 MapReduce

Map-reduce frame work has two phases, Map phase and reduce phase. These two phases are connected by a data shuffle. On each mapper (worker) node, the map phase loads local data processes it by a user-defined function. That UDF is state-less and takes as input a chunk of local data and output a list of key-value pairs. At start of reduce phase, keys are shuffled between reducer (worker) nodes, and data is pulled by the reducer. The reducer takes that list values associated with the current key and applies another state-less UDF.

Apache Hadoop is the open source version of MapReduce. Hadoop/MapReduce is meant for long-runtime batch processing. Writing intermediate results to disk to be read later.

Apache Hadoop platform

1. Applications : Hive, Pig

2. Data Processing Farmeworks : MapReduce, Spark, Flink, Storm, Tez

3. Resource Mangagement : Yarn, Mesos

4. Storage Mangagement : HDFS, HBase, ect.

1.6.2 Spark

Spark took the same map-phase reduce-phase architecture as MapReduce and said, let's store as much of the intermediate data in memory to speed up processing. Disk I/O is very slow, so storing data in memory is much faster. This meant that for reasonable sized datasets, spark can answer queries in hours, where MapReduce took days.

Spark has become a much more popular data processing platform than Hadoop. Spark can support a wider range of data-scenarios including SparkMLlib, GraphX, Spark SQL, and spark streaming because it is much faster. Data is stored in RDD's (Resilient distributed datasets) that are stored across memory and spills to disk. Computation is done in directed acyclic graph of map and reduce phases like Apache Tez.

The idea of storing data in memory as much as possible is an old idea of distributed shared memory (DSM). Distributed shared memory was to share memory across machines to create a larger logical pool of memory. Despite DSM's being well liked in the research community, they were not used in real systems.

Question: Why are RDD's "resilient"

Answer: RDD's can be reconstructed from how they were derived. Rolling back to the most recent parents of the lost RDD, we redo computation to reconstruct the lost RDD. In that way, RDD's are resilient to node failures.