Distributed Shared Memory

Traditional distributed systems area.

Overview of Area

"A Comprehensive Bibliography of Distributed Shared Memory" by Eskicioglu (U. of New Orleans). Operating Systems Review. 1995.

"Distributed Shared Memory: Concepts and Systems" by Protic, Tomasevic and Milutinovic (U. of Belgrade). IEEE Parallel & Distributed Technology. Summer 1996. Slides:
`http://www.cs.wpi.edu/~cs535/s08/protic.pdf`

Lots of additional work.

Linda Systems

One classic shared data approach taken by Linda system and successors. Supports sharing and coordination.

David Gelernter and Nicholas Carriero, Yale

Immutable data approach typified by Linda-type systems, which use a *tuple space*. Tuples consist of a sequence of one or more typed data fields such as `<"fred", 1958>`, `<4, 3.2, "xyz">`.

Operations:

- read, take—block until a tuple is available that match tuple criteria such as `<String, 1958>`. *take* removes tuple from space.
- write—adds tuple to space

Example:

```
<s,count> = myTS.take(<"counter", integer>);
myTS.write(<"counter", count+1>
```

Jini

Set of distributed system services in Java. Originally developed by Sun, but now an incubator project called River in Apache.

Uses a Lookup Service within a *community*, which might be a LAN.

Key concepts of JINI:

1. Discovery
    - Multicast Request Protocol used by an application to find a Locate Service.
    - Multicast Announcement Protocol to multicast the availability of a service to Locate Service.
    - Unicast Discovery Protocol when location of service is known—accessed via URL such as `jini://jiniserver.wpi.edu`
2. Join—register a service with Lookup Service
3. Lookup—find a service with Lookup Service
4. Leasing—servers must periodically re-register services with Lookup Service.
5. Remote Events—register an interest in an event with Lookup Service.
6. Transactions—interface to implement transactions
7. Coordination

JavaSpaces

Shared data space.

Service that can be implemented in Jini. See Fig 12-15.

A Linda-like shared dataspace. Can be used for coordination of processes.

Tuples are typed references to Java objects (name, value pairs). Tuple contents are marshaled when stored into JavaSpace.

Can have multiple *instances* of a tuple. See Fig. 12-14. Can have multiple versions of a tuple.

Need to supply a *template* when reading a tuple. Only instances matching the template are returned. Can have the client block until a matching tuple becomes available.

Can use events in combination with JavaSpaces.

JavaSpaces Application

Can be used for different purposes. For example, can build a ComputeFarm with it where:

1. "task tuples" are created,
2. which are consumed by "workers", which then create "result tuples",
3. which are consumed by "result harvesters".

Similar type approach as MapReduce idea, although not the same low-level coordination between data and computation.