Distributed System (DS)

- More than one processor
- More than one task (or process)
- More than one protected memory area
- Communication by message passing
- Communication not free and include delay
- Partly common goal solved by cooperation

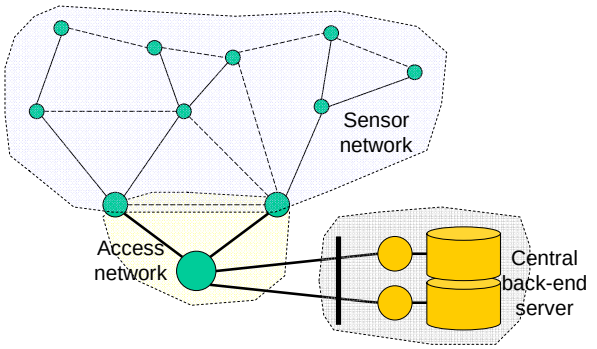
Design Choices for DS

- Central control is partly replaced by distributed and decentralised (more local and autonomous) control
- However, some common global state awareness and control is still mostly necessary in most systems
- Central shared memory or distributed local memories or a combination
- Communication based on message (value) passing and/or (reference to) shared data in memory
- Communication via network links or memory
- Can be based on a few complex and/or many simple processors (coarse or fine grain)

Distributed Systems (DS)

- Parallel
  - Lockstep actions on synchronized data streams
  - Single instruction multiple data (SIMD)
- Concurrent
  - Actions performed in any order
  - Partial independence (via buffering)
  - Multiple Instruction Multiple Data (MIMD)
- Distributed
  - Looser coupling but coordinated cooperation between tasks
- Decentralized
  - Very loose coupling and no central unit for coordination
  - Autonomous reflecting individual behaviour
  - Common behaviour rules and goals is used to get cooperation and coordination

Sensor Network



Sensor Network

Parallel Computing

Flynn's Classification of computer architectures

S = Single

M = Multiple

I = Instruction (control thread)

D = Data (processing path)

SISD = Single Instruction Single Data

SIMD = Single Instruction Multiple Data

MISD = Multiple Instruction Single Data

MIMD = Multiple Instruction Multiple Data

Model of Multi Processor System

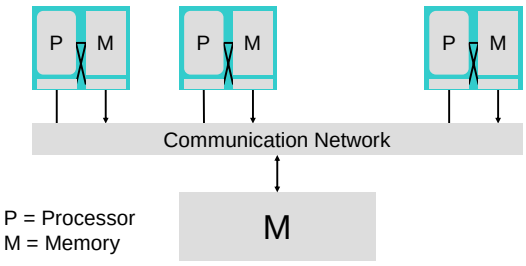
- More than one processor
- Shared memory (supported by cache memory)
- Communication between tasks executed by different processors is local and efficient
- Frequent but short distance communication
- Tight coupling via shared memory or switched network
- Shared memory becomes bottleneck (limit no processors < 4-8)

Varying Application Requirements

- High-throughput or computing capacity needs?
- Quick response time?
- Fault-tolerant and highly available?
- High scalability to customer needs?
- Physically distributed (sensors/actuators)?
- Fixed or highly varying load from application environment or users?

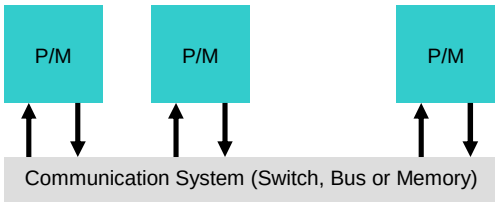
Physical (HW) System Structure

Processor-Memory-Switch (PMS) structure



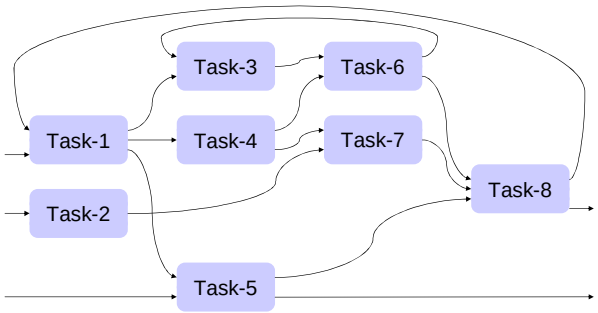
Physical (HW) System Structure

Processor-Memory-Switch (PMS) structure



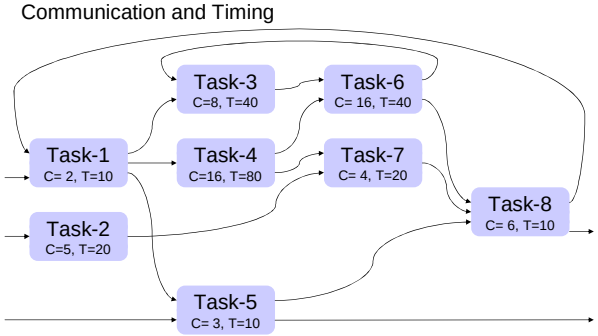
P/M = Processor or memory or both processor and memory

Logical (SW) System Structure

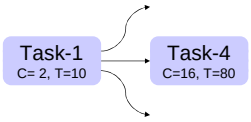


Task Dependency Structure

Communication and Timing



Different activation periods



Assumption:
A sending task (acting as a source) will produce a message in every period and send it individually via a buffer or aggregated in a package according to the needs of the receiving task and its execution period.

For example:
Task-4 will 1) use the most recent message or 2) get eight aggregated messages from Task-1 when activated.

Distributed programming

Program tasks can be described using a process algebra:

A ; B means A is followed by B as two sequentially executed tasks

A | B means that either A or B is executed

A // B means A is executed concurrently with B

A || B means A is executed simultaneous (parallel) with B

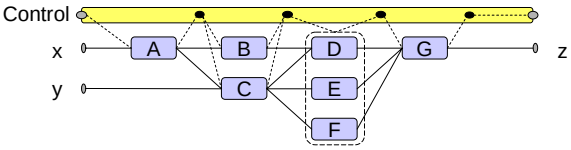
x : A means A is executed after that the event x has occurred.

The dot (.) means end of program

Distributed programming, cont.

Concurrent tasks with execution dependencies:

```
P(x,y,z)  =  A(x,a);
             B(a,b) | C(y,a,c);
             D(b,c,d) // E(c,e) // F(c,f);
             G(d,e,f,z).
```



Handling of shared data

- Data representing important system or resource state must often be shared between several concurrent jobs (task instances)
- A database manager can simplify the handling of such shared information in a coordinated way
- Mechanisms to implement synchronized transactions over distributed data storage locations may also be needed

Concurrent (shared data) objects

Concurrent objects are:

- data stored in memory shared by more than one task
- can be scalar data but mostly composed data structures that contain more than one memory position or value
- manipulated via operations associated with the object

A concurrent object is a:

- specialized data structure (record, queue, list, tree, hash table, …)
- memory pool

Concurrent objects are supported by:

- database handler, transaction handler, …
- transactional memory (TM), Software TM (STM), …
- tuple space, black board, …
- protocol for reservation, release, queries and updates, …

Concurrent objects, cont.

- **By implementing coordinated managed access** to such a data structure one can allow more than one task to use the data structure concurrently
- For example, one can do atomic read and/or write operations in different parts of such a structure at the same time without conflicts

A concurrent object is:

- **Non-blocking** if some task will complete an operation in a finite number of steps (the processor is not blocked and thus utilized efficiently)
- **Wait-free** if each task will complete an operation in a finite number of steps (no task starvation or live-lock)

Transactional Memory (TM)

Language or OS services for lock(X) and unlock(X)

Lock(X)

… operations on data reserved by lock …

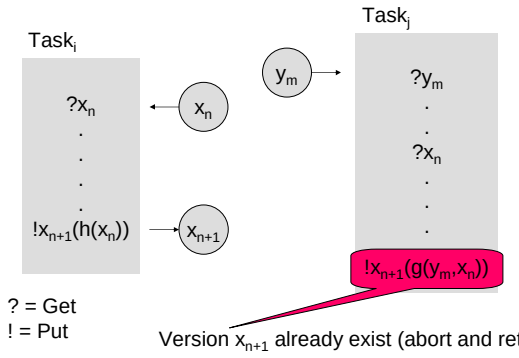
Unlock(X)

Replaced by

Atomic(…operations on object(X_n)…resulting in version n+1)

Advantage: declarative specification
implemented by OS keeping track of and avoiding object version conflicts

Task transaction conflict detection



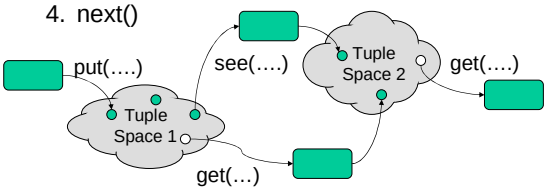
? = Get
! = Put

Version x_{n+1} already exist (abort and retry)

Task cooperation via Tuple Space

Methods for task interaction via Tuple Space:

1. put()
2. get()
3. see()
4. next()



Tuples

A tuple is a marked or tagged data record

For example:

get(CAR, x, Green) will return a tuple

<CAR, 33, Green>

if this tuple is the next item in the tuple space that fulfil the request

Tuple Space

A tuple storage technique intended to simplify task:

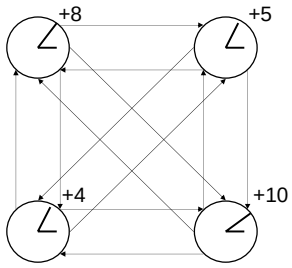
- Integration
- Cooperation
- Communication
- Coordination
- Data and object sharing / persistency
- Activation by data entry and event leasing
- Transactions
  - Atomic
  - Nested
  - Distributed
- Decoupling (acting as a buffer between producer and consumer)

Synchronized time

- All nodes of a distributed system must have a shared notion of the current (clock) time in the system
- To implement a shared view of time, the view of clock time must be agreed upon between nodes
- This using:
  - protocol for exchange of time data
  - mechanisms for filtering and calculation of a shared (weighted average) system clock time

Clock synchronization

If clock adjustment messages are sent in ring one can get an oscillating pattern: 6.5, 8.25, 6.12, 7, 6, 8, 6, 7, 6, 8, 6, 7, 6 …



$$\frac{5+8+10+4}{4} = 6{,}75$$

Would a random pair wise adjustment scheme work better?

Synchronized time, cont.

- The clock system is often based on a central time reference complemented with, less accurate, local clocks that is updated by communication with the central clock
- The clock system can also be totally decentralized and use the local clock time of all non faulty nodes to calculate a common (average) view of the overall system clock time
- Clocks that drift or temporary deviate too much (i.e. outliers) are discarded and possibly considered faulty
- The calculated clock time can also be given a suitable amount of momentum (higher weight to old values) to be less sensitive to noise and oscillating disturbances

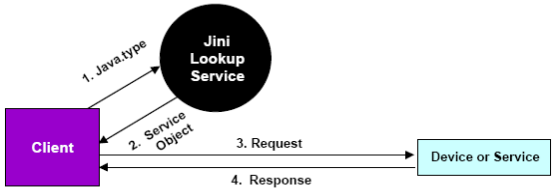
Software in distributed system

- To have software components and their services distributed over several processors requires that they can be made known to each other
- Announcement (publishing) of component services can be made statically or dynamically
- In the dynamic case the name (or other identifier) of services, components or their interfaces are announced (published) to a register where they can be found by client components

Distributed software, cont.

- Some services are passive and respond
  - directly to a remote procedure call
  - delayed using a call back function
- Other service can be more active periodic or aperiodic “agents” or “producers” that one must connect to, to get results from
- Such service producer connections are sometimes called (event) subscriptions

JINI



Source: Bill Day, SUN Microsystems

Distributed software, cont.

- To cope with real-time programming problems specialized programming languages, middleware (MW) and operating system (OS) mechanisms are developed
- These provide support that simplify the handling of resource conflicts and reduce the risk for locking code

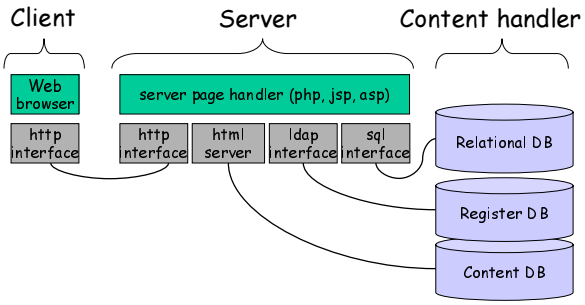
Distributed software, cont.

- For software that deals with signal processing there are specialized languages that simplifies the application of repetitive operations on streaming data

Distributed software, cont.

- Programming models designed for embedded real-time applications support components (called blocks, actors or tasks) to communicate via ports, connectors or channels
- Ports, connectors or channels can decouple the components or tasks (and prevent them from knowing too much about each other)

Presentation, service logic and content handling



Local and Distributed Tasks/Objects

- A program made to access an object outside the local address space, perhaps on a different machine, must consider failures, indeterminacy, and concurrency constraints
- Programmers can not be entirely protected from reality and thus must be aware of the problems that sooner or latter can arise

Distributed Computing Environment

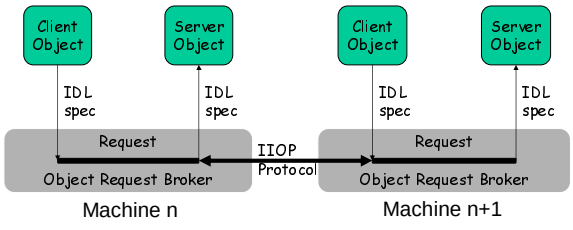
Open Software Foundation (OSF), 1992

DCE includes definitions and support for:

- Threads
- Remote Procedure Call (RPC)
- Distributed Time Service
- Name Service
- Distributed Service

CORBA

Common Object Request Broker Architecture



Interoperability means

- IDL, XML, ASN.1, …
- RTP, SIP, SMTP, HTTP, FTP, SNMP, …
- TCP, UDP, SCTP, …
- IP
- Profibus, CAN, Token Ring IEEE 802.5, Ethernet IEEE 802.3, WLAN IEEE 802.11, PCI, RapidIO, Infiniband, I2C, USB, IEEE 1394, Bluetooth, IEEE 811.16.4, RS232, RS422, RS485, …

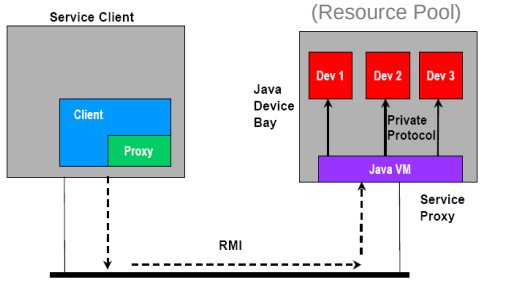
Interoperability standards, examples

- Application and presentation level, e.g. ASN.1, IDL and XML based information exchange
- Protocols for installation, registration, discovery and distribution e.g., JINI, UDDI, SIP, OSGi
- Object access CORBA, DCOM, .NET, SOAP

ASN.1 = Abstract Syntax Notation One
IDL = Interface Definition Language (IIOP)
OSGi = Open Service Gateway initiative
SOAP = Simple Object Access Protocol (XML, HTTP)
WSDL = Web Services Description Language
UDDI = Universal Description Discovery and Integration
SIP = Session Initiation Protocol

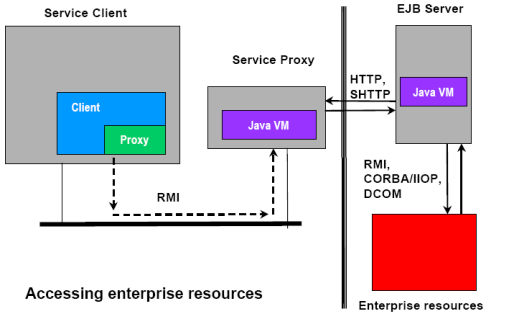
JINI

Java dynamic networking technique for building of distributed systems that are highly adaptive to change



Source: Bill Day, SUN Microsystems

JINI



Source: Bill Day, SUN Microsystems

Dynamic "Ad hoc" couplings

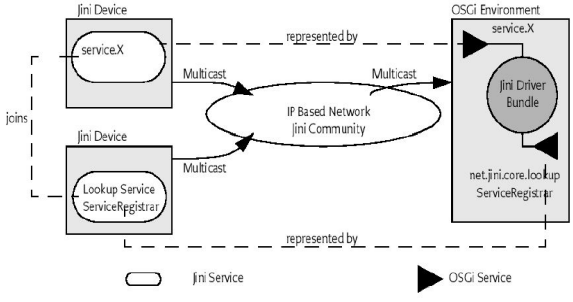
Wireless connections enable:

- temporary, dynamic and mobile meetings
- sharing of resources (client-to-server or peer-to-peer)

Implying:

- new resource conflicts to solve
- more varying work (environment) conditions

JINI services brooked via OSGi



Source: Richard van der Laan, LUMNIS