Cluster Computing

DR. SHAKTI MISHRA

History

The first inspiration for cluster computing was developed in the 1960s by IBM as an alternative of linking large mainframes to provide a more cost-effective form of commercial parallelism.

Cluster computing did not gain momentum until the convergence of four important trends in the 1980s:

- high-performance microprocessors,
- high-speed networks, and
- standard tools for high performance distributed computing.
- increasing need of computing power for computational science and commercial applications coupled with the high cost and low accessibility of traditional supercomputers.

Cluster Architecture

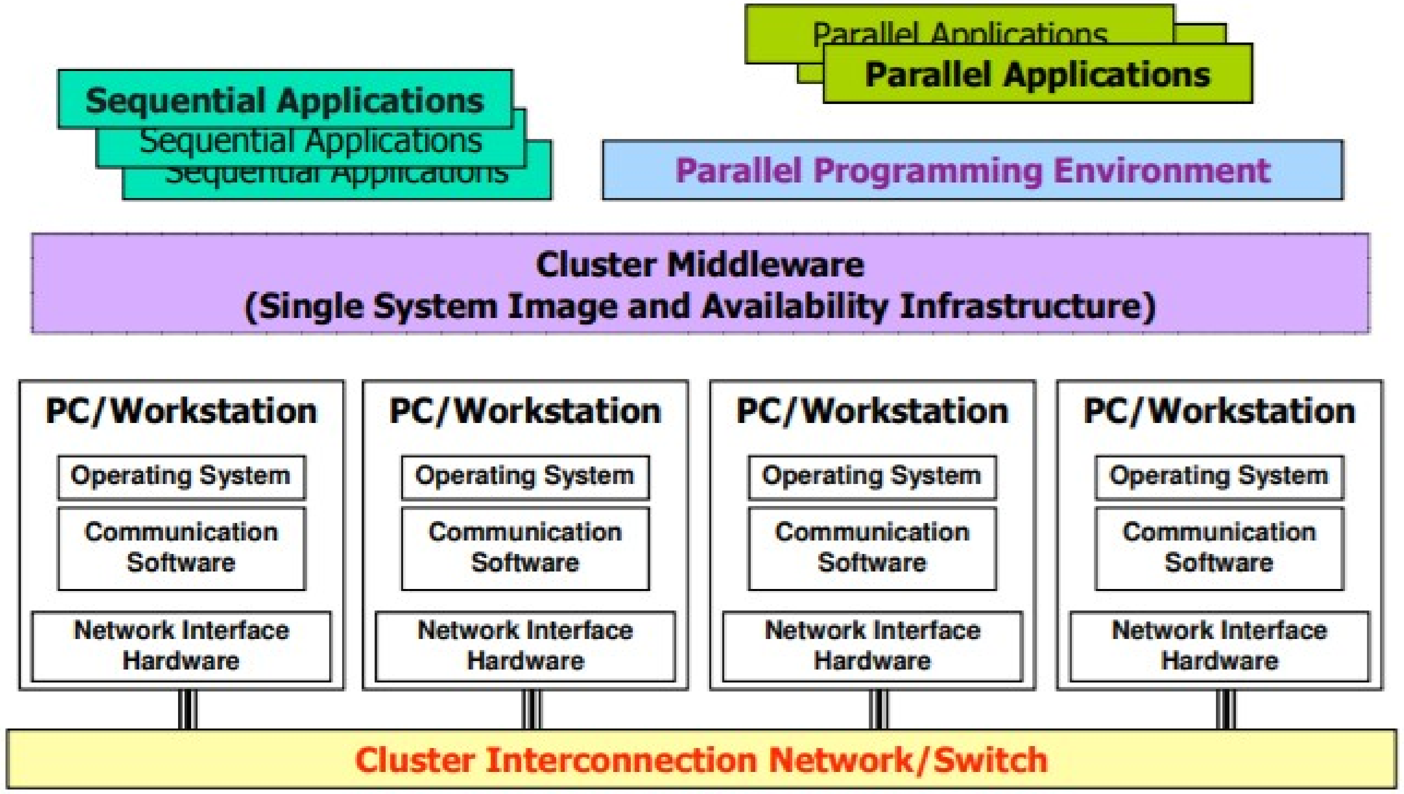


Figure 1. Cluster architecture (R. Buyya [1]).

Advantages

The emergence of cluster platforms was driven by a number of academic projects, such as Beowulf [2], Berkeley NOW [3], and HPVM [4]

Advantages of Cluster Computing:

- low-entry costs to access supercomputing-level performance,
- the ability to track technologies,
- incrementally upgradeable system,
- open source development platforms, and
- vendor independence

Cluster Computing: Definition & Components

A cluster is a type of parallel or distributed computer system, which consists of a collection of interconnected stand-alone computers working together as a single integrated computing resource.

The key components of a cluster include multiple standalone computers (PCs, Workstations, or SMPs), operating systems, high-performance interconnects, middleware, parallel programming environments, and applications.

Cluster: Interconnection Technologies

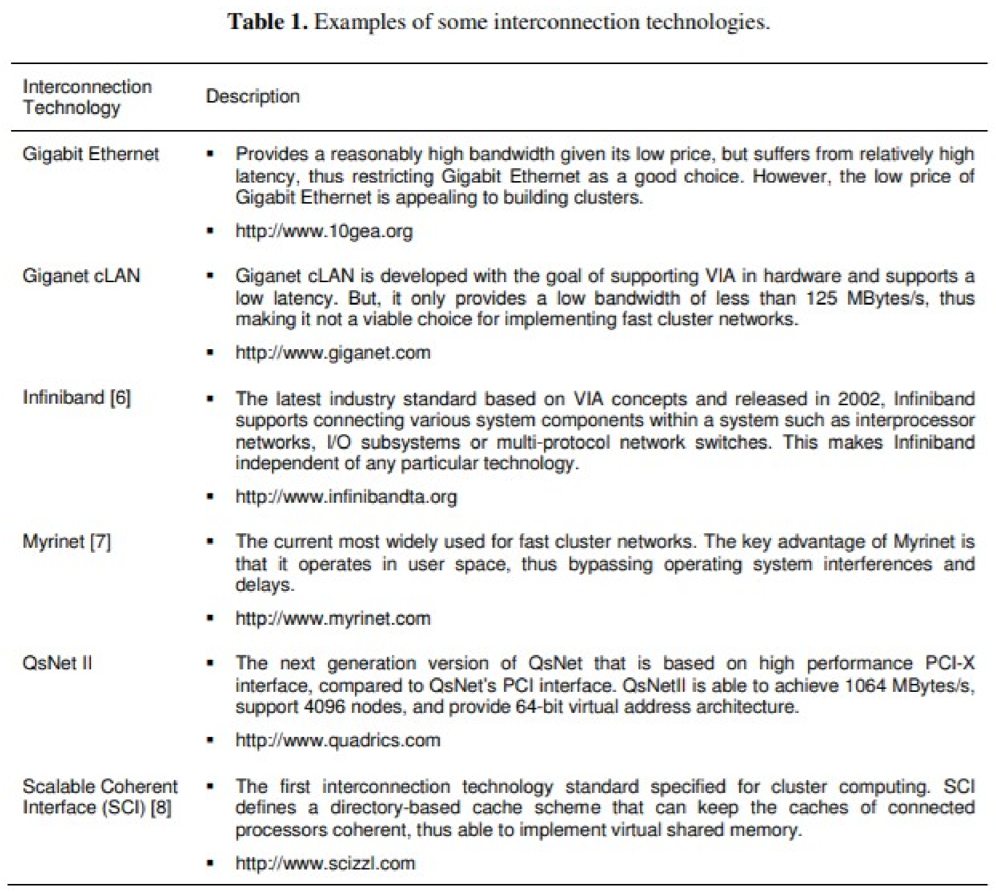


Table 1. Examples of some interconnection technologies.

Interconnection Technology	Description
Gigabit Ethernet	▪ Provides a reasonably high bandwidth given its low price, but suffers from relatively high latency, thus restricting Gigabit Ethernet as a good choice. However, the low price of Gigabit Ethernet is appealing to building clusters. ▪ http://www.10gea.org
Giganet cLAN	▪ Giganet cLAN is developed with the goal of supporting VIA in hardware and supports a low latency. But, it only provides a low bandwidth of less than 125 MBytes/s, thus making it not a viable choice for implementing fast cluster networks. ▪ http://www.giganet.com
Infiniband [6]	▪ The latest industry standard based on VIA concepts and released in 2002, Infiniband supports connecting various system components within a system such as interprocessor networks, I/O subsystems or multi-protocol network switches. This makes Infiniband independent of any particular technology. ▪ http://www.infinibandta.org
Myrinet [7]	▪ The current most widely used for fast cluster networks. The key advantage of Myrinet is that it operates in user space, thus bypassing operating system interferences and delays. ▪ http://www.myrinet.com
QsNet II	▪ The next generation version of QsNet that is based on high performance PCI-X interface, compared to QsNet's PCI interface. QsNetII is able to achieve 1064 MBytes/s, support 4096 nodes, and provide 64-bit virtual address architecture. ▪ http://www.quadrics.com
Scalable Coherent Interface (SCI) [8]	▪ The first interconnection technology standard specified for cluster computing. SCI defines a directory-based cache scheme that can keep the caches of connected processors coherent, thus able to implement virtual shared memory. ▪ http://www.scizzl.com

Single System Image (SSI)

Represents the view of a distributed system as a single unified computing resource.

Provides better usability for the users as it hides the complexities of the underlying distributed and heterogeneous nature of clusters from them.

SSI can be established through one or several mechanisms implemented at various levels of abstraction in the cluster architecture: hardware, operating system, middleware, and applications

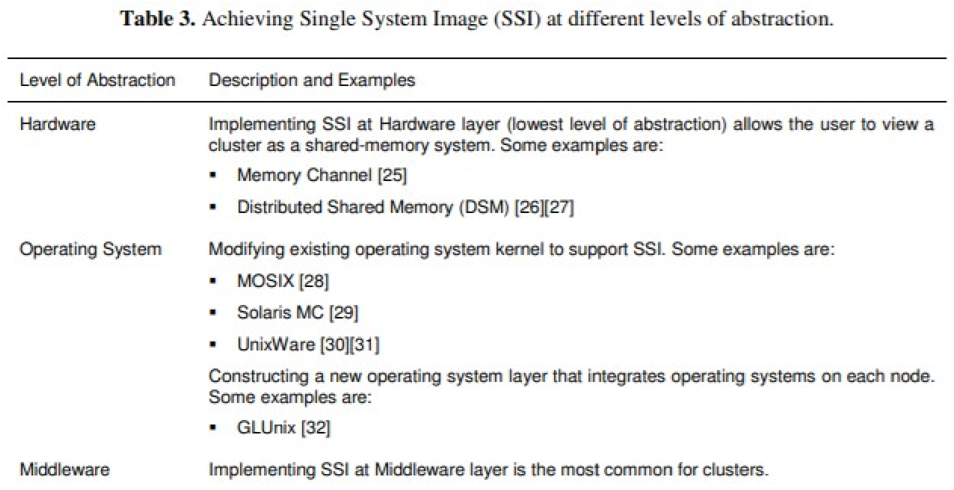


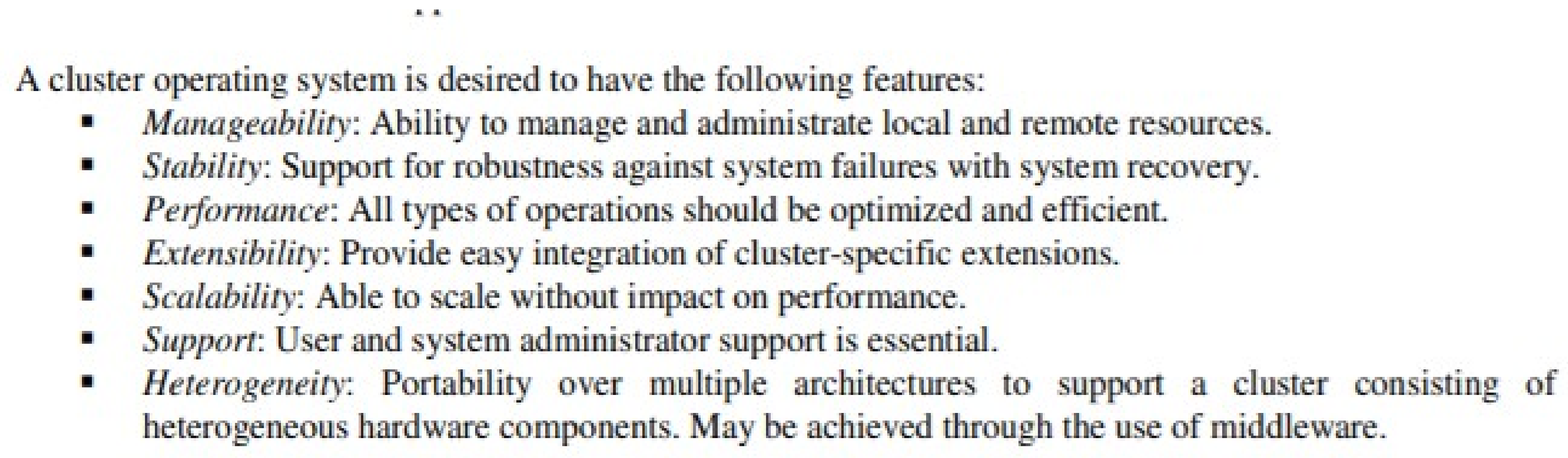
Table 3. Achieving Single System Image (SSI) at different levels of abstraction.

Level of Abstraction	Description and Examples
Hardware	Implementing SSI at Hardware layer (lowest level of abstraction) allows the user to view a cluster as a shared-memory system. Some examples are: ▪ Memory Channel [25] ▪ Distributed Shared Memory (DSM) [26][27]
Operating System	Modifying existing operating system kernel to support SSI. Some examples are: ▪ MOSIX [28] ▪ Solaris MC [29] ▪ UnixWare [30][31] Constructing a new operating system layer that integrates operating systems on each node. Some examples are: ▪ GLUnix [32]
Middleware	Implementing SSI at Middleware layer is the most common for clusters.

SSI at OS Level

The operating system in each of the cluster nodes provides the fundamental system support for the combined operation of the cluster.

The operating system provides services such as protection boundaries, process/thread coordination, inter-process communication, and device handling, thus creating a high-level software interface for user applications



A cluster operating system is desired to have the following features:

- *Manageability*: Ability to manage and administrate local and remote resources.
- *Stability*: Support for robustness against system failures with system recovery.
- *Performance*: All types of operations should be optimized and efficient.
- *Extensibility*: Provide easy integration of cluster-specific extensions.
- *Scalability*: Able to scale without impact on performance.
- *Support*: User and system administrator support is essential.
- *Heterogeneity*: Portability over multiple architectures to support a cluster consisting of heterogeneous hardware components. May be achieved through the use of middleware.

Resource Management System (RMS) Middleware

A cluster resource management system (RMS) acts as a cluster middleware that implements the SSI [24] for a cluster of machines.

It enables users to execute jobs on the cluster without the need to understand the complexities of the underlying cluster architecture.

A RMS manages the cluster through four major branches, namely: resource management, job queuing, job scheduling, and job management.

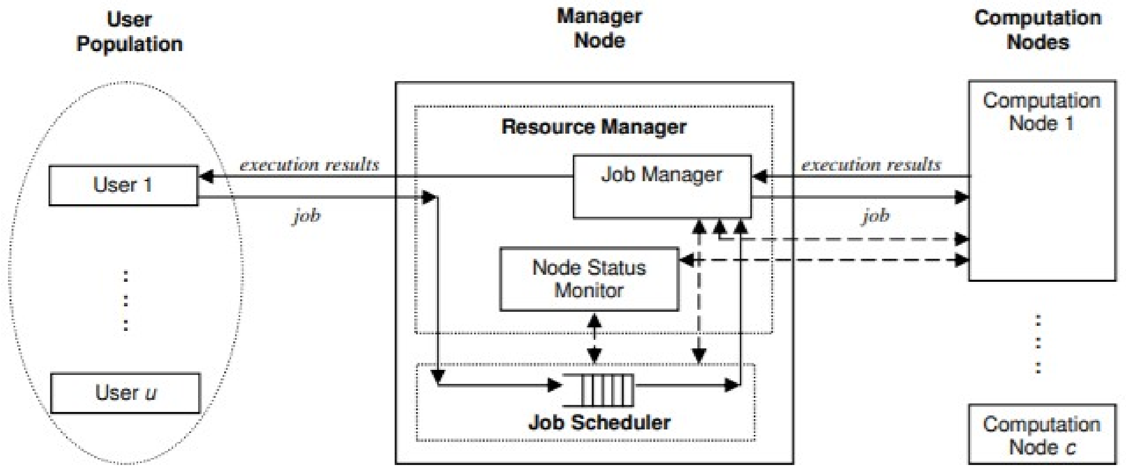


Figure 8. Cluster RMS architecture.

RMS Middleware

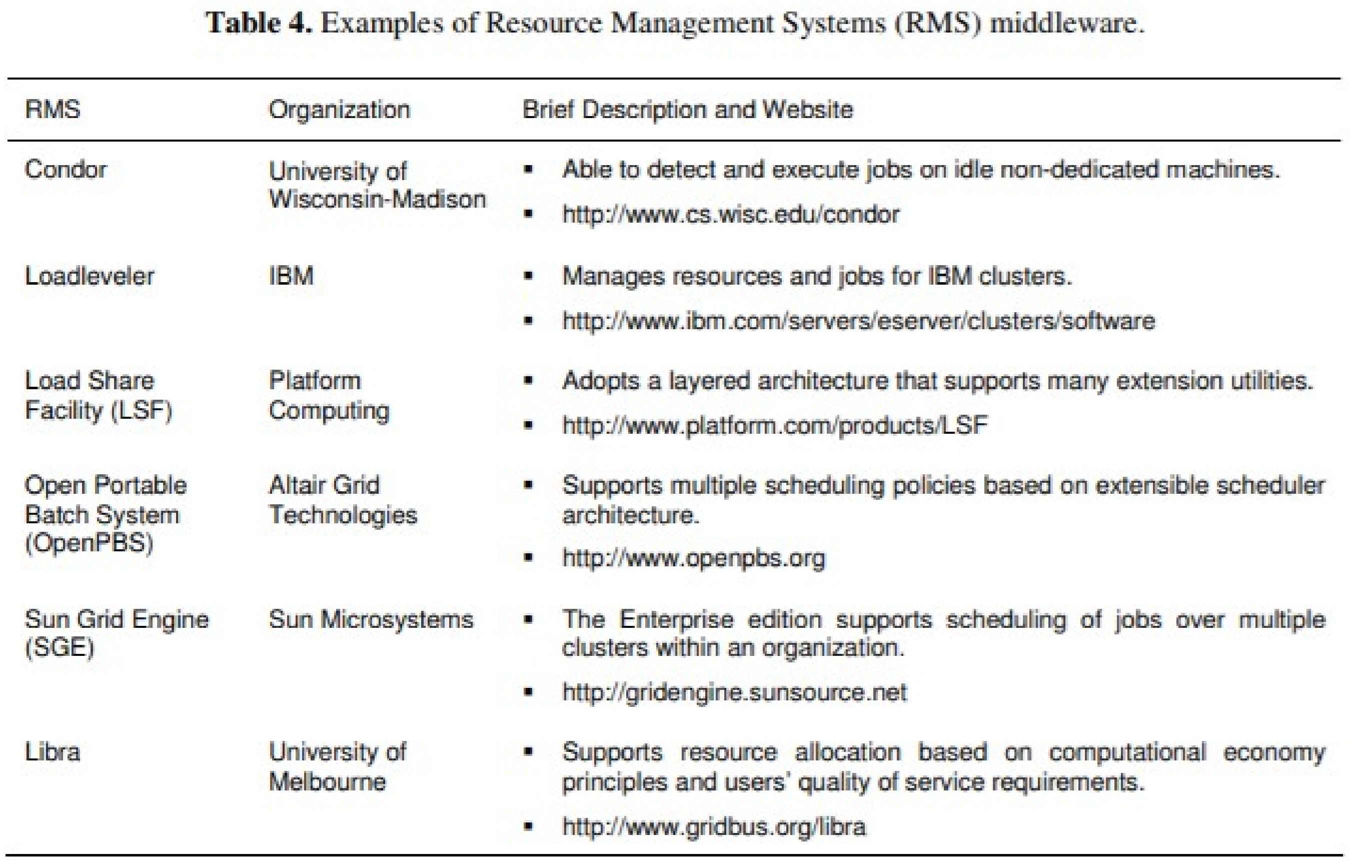


Table 4. Examples of Resource Management Systems (RMS) middleware.

RMS	Organization	Brief Description and Website
Condor	University of Wisconsin-Madison	▪ Able to detect and execute jobs on idle non-dedicated machines. ▪ http://www.cs.wisc.edu/condor
Loadleveler	IBM	▪ Manages resources and jobs for IBM clusters. ▪ http://www.ibm.com/servers/eserver/clusters/software
Load Share Facility (LSF)	Platform Computing	▪ Adopts a layered architecture that supports many extension utilities. ▪ http://www.platform.com/products/LSF
Open Portable Batch System (OpenPBS)	Altair Grid Technologies	▪ Supports multiple scheduling policies based on extensible scheduler architecture. ▪ http://www.openpbs.org
Sun Grid Engine (SGE)	Sun Microsystems	▪ The Enterprise edition supports scheduling of jobs over multiple clusters within an organization. ▪ http://gridengine.sunsource.net
Libra	University of Melbourne	▪ Supports resource allocation based on computational economy principles and users' quality of service requirements. ▪ http://www.gridbus.org/libra

Cluster Programming Models

Cluster computing programming models have traditionally been divided into categories based on the relationship of programs to the data the programs operate on:

The Single-Instruction, Single-Data (SISD) model defines the traditional von Neumann computer.

- each processor executes the same program

Multiple-Instruction, Multiple-Data (MIMD) machines.

In the Single-Instruction, Multiple Data (SIMD) model.

- multiple programs operate on the same data.

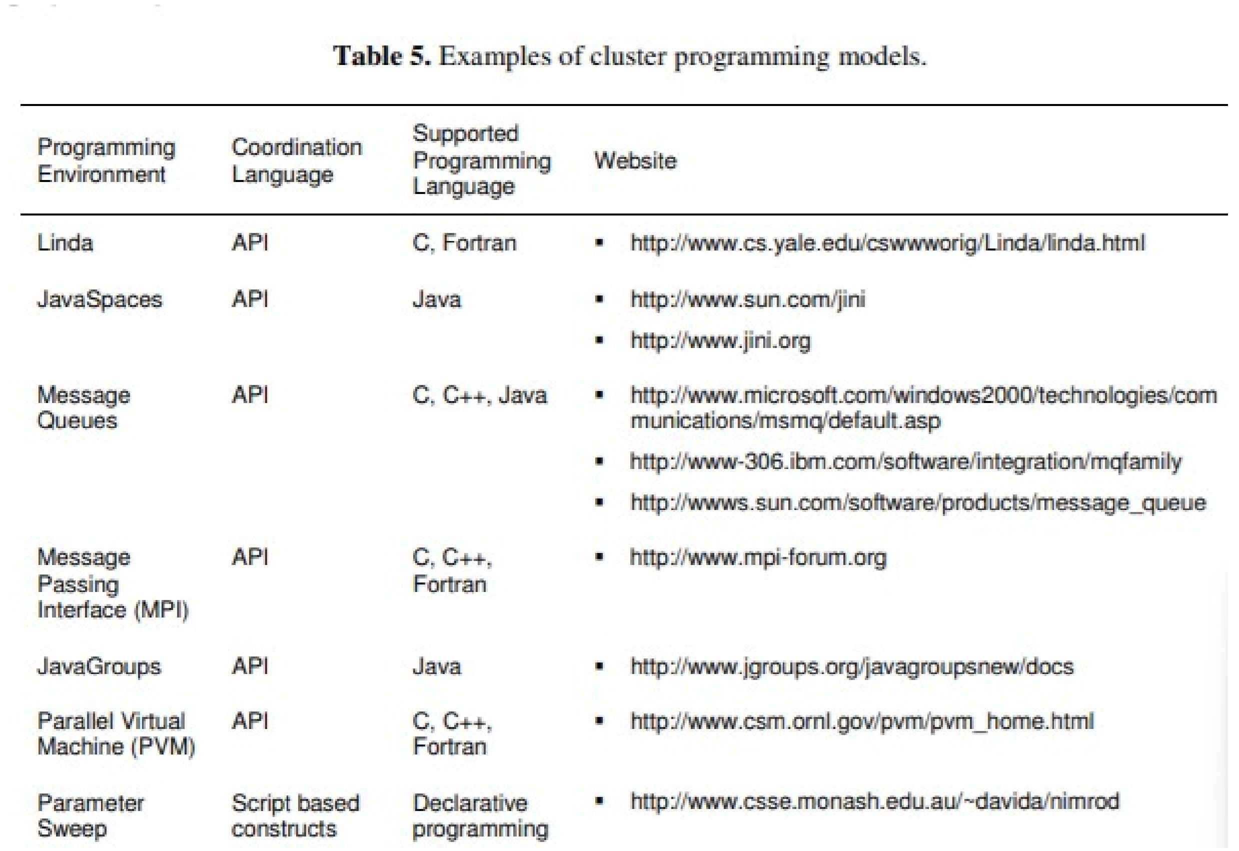


Table 5. Examples of cluster programming models.

Programming Environment	Coordination Language	Supported Programming Language	Website
Linda	API	C, Fortran	▪ http://www.cs.yale.edu/cswwworig/Linda/linda.html
JavaSpaces	API	Java	▪ http://www.sun.com/jini ▪ http://www.jini.org
Message Queues	API	C, C++, Java	▪ http://www.microsoft.com/windows2000/technologies/communications/msmq/default.asp ▪ http://www-306.ibm.com/software/integration/mqfamily ▪ http://wwws.sun.com/software/products/message_queue
Message Passing Interface (MPI)	API	C, C++, Fortran	▪ http://www.mpi-forum.org
JavaGroups	API	Java	▪ http://www.jgroups.org/javagroupsnew/docs
Parallel Virtual Machine (PVM)	API	C, C++, Fortran	▪ http://www.csm.ornl.gov/pvm/pvm_home.html
Parameter Sweep	Script based constructs	Declarative programming	▪ http://www.csse.monash.edu.au/~davida/nimrod

Cluster Programming Models

Programming models can also be categorized on how they exploit a cluster's inherent parallelism.

cluster computing programming models can roughly be divided in two categories:

The first category of models allow a serial (non-parallel) application to take advantage of a cluster's parallelism.

The second category of programming models aid in the explicit parallelization of a program.

In SPPS (serial program, parallel subsystem), many instances of a serial program are distributed on a cluster.

A parallel subsystem provides input to each serial program instance, and captures output from those programs, delivering that output to users.

Because there are multiple programs on the clusters, operating on multiple data, SPPS is a form of MIMD.

Cluster Programming Models

When many instances of a serial program operating in parallel, those instances must coordinate work through a shared cluster resource, such as distributed shared memory or a message passing infrastructure.

The primitive operations that coordinate the work of concurrently executing serial programs on a cluster define a coordination language.

A coordination language is often described in terms of an Application Programming Interface (API) to a parallel subsystem

Example: LINDA

The Linda tuple-space system exploits distributed shared memory to facilitate the parallel execution of a serial program on a cluster.

Linda defines primitive operations on a shared memory resource, allowing data items – “tuples” – to be written to that memory, read from shared memory, and deleted from shared memory.

A tuple is similar to a database relation.

A serial process with access to the shared memory writes data items to memory, marking each item with an attribute indicating that that item requires processing.

Another process awaiting newly arriving tuples removes such an item from the shared memory, performs computations on that data item, and deposits the results into the shared memory.

The original submitter of the job then collects all the results from the tuple-space.

Each process operating on a tuple-space is typically a serial program, and the Linda system facilitates the concurrent execution of many such program

Example: JAVASPACES

JavaSpaces is an object-oriented Linda system that takes advantage of Java’s platform-independent code execution and mobile code facility.

Mobile code allows not just data, but also code to move from one cluster node to another at runtime.

A master node runs a JavaSpace process, providing a shared memory resource to other cluster nodes that act as workers.

When a worker removes a job request from the shared JavaSpace, the operating codes for that job dynamically download to that worker. The worker executes that downloaded code and places the output of that execution into the JavaSpace.

JavaSpaces, therefore, facilitates the automatic runtime distribution of code to cluster nodes.

JavaSpaces also provides optional transactional access to the shared memory resource, which is especially helpful in the case of very large clusters with frequent node failures

Case Study: Google Search Engine

Google uses cluster computing to meet the huge quantity of worldwide search requests that comprise of a peak of thousands of queries per second.

A single Google query needs to use at least tens of billions of processing cycles and access a few hundred megabytes of data to return satisfactory search result.

Google uses cluster computing as its solution to the high demand of system resources since clusters have better price-performance ratios than alternative high-performance computing platforms and use less electrical power.

Google focuses on 2 important design factors: reliability and request throughput.

Case Study: Google Search Engine

Google is able to achieve reliability at the software level so that a reliable computing infrastructure can be constructed on clusters of 15,000 commodity PCs distributed worldwide.

The services for Google are also replicated across multiple machines in the clusters to provide the necessary availability.

Google maximizes overall request throughput by performing parallel execution of individual search requests. This means that more search requests can be completed within a specific time interval.

Case Study: Google Search Engine

A typical Google search consists of the following operations:

1. An Internet user enters a query at the Google webpage.

2. The web browser searches for the Internet Protocol (IP) address via the www.google.com Domain Name Server (DNS).

3. Google uses a DNS-based load balancing system that maps the query to a cluster that is geographically nearest to the user so as to minimize network communication delay time. The IP address of the selected cluster is returned.

4. The web browser then sends the search request in Hypertext Transport Protocol (HTTP) format to the selected cluster at the specified IP address.

5. The selected cluster then processes the query locally.

6. A hardware-based load balancer in the cluster monitors the available set of Google Web Servers (GWSs) in the cluster and distributes the requests evenly within the cluster.

7. A GWS machine receives the request, coordinates the query execution and sends the search result back to the user’s browser.

Case Study: Google Search Engine

The first phase of query execution involves index servers consulting an inverted index that match each query keyword to a matching list of documents.

- Relevance scores are also computed for matching documents so that the search result returned to the user is ordered by score.

In the second phase, document servers fetch each document from disk to extract the title and the keyword-in-context portion of the document.

In addition to the 2 phases, the GWS also activates the spell checker and the ad server.

The spell checker verifies that the spelling of the query keywords is correct, while the ad server generate advertisements that relate to the query and may therefore interest the user

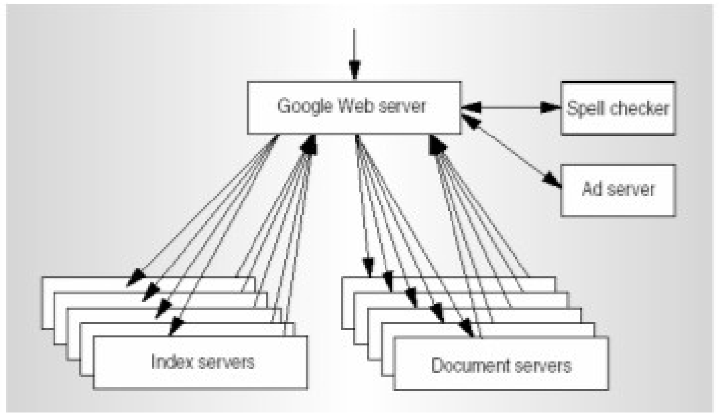


Figure 9. Google query-serving architecture (L. A. Barroso, et al. [68]).