|  |
| --- |
|  |
|  |
|  |

rectangle

rectangle

Table of Content

Contents

[A.](#_heading=h.gjdgxs) Pre-Knowledge 3

[I.](#_heading=h.30j0zll) Message Passing Interface (MPI) 3

[1. Overview 3](#_heading=h.1fob9te)

[2. Definition 3](#_heading=h.3znysh7)

[3. Advantages of MPI 4](#_heading=h.2et92p0)

[4. Mechanism 4](#_heading=h.tyjcwt)

[5. Noticeable function 6](#_heading=h.3dy6vkm)

[II.](#_heading=h.1t3h5sf) Peer-to-peer System (P2P) 8

[1. Definition & Key value 8](#_heading=h.4d34og8)

[2. Some terminologies 8](#_heading=h.2s8eyo1)

[3. Taxonomy 10](#_heading=h.17dp8vu)

[4. P2P System 10](#_heading=h.3rdcrjn)

[5. P2P Application 11](#_heading=h.26in1rg)

[6. P2P Models and Algorithm 12](#_heading=h.lnxbz9)

[8. Characteristic 15](#_heading=h.35nkun2)

[9. Security 23](#_heading=h.1ksv4uv)

[10. Strengths and Weaknesses 24](#_heading=h.44sinio)

[III.](#_heading=h.2jxsxqh) Hybrid Centralization and P2P System 26

[1. Hybrid Procurement Systems 26](#_heading=h.z337ya)

[2. Hybrid P2P 26](#_heading=h.3j2qqm3)

[3. Requirements 27](#_heading=h.1y810tw)

[4. Implementation 27](#_heading=h.4i7ojhp)

[B.](#_heading=h.2xcytpi) Load Testing 29

[1. User experience under load test 29](#_heading=h.1ci93xb)

[**Stress testing** 29](#_heading=h.3whwml4)

[2. Three important response time values: 29](#_heading=h.2bn6wsx)

[3. Creating a Benchmark Test Plan 30](#_heading=h.qsh70q)

[4. Why should we use MPI 31](#_heading=h.3as4poj)

[C.](#_heading=h.1pxezwc) Further Details 32

[I.](#_heading=h.49x2ik5) Members and Duties 32

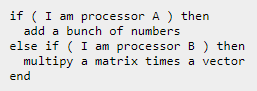
[II.](#_heading=h.3fwokq0) Research 32

# Pre-Knowledge

# Message Passing Interface (MPI)

### 1. Overview

A remarkable feature of MPI is that the user writes a single program which runs on all the computers. However, because each computer is assigned a unique identifying number, it is possible different actions to occur on different machines, even though they run the same program:



Another feature of MPI is that the data stored on each computer is entirely separate from that stored on other computers. If one computer needs data from another, or wants to send a particular value to all the other computers, it must explicitly call the appropriate library routine requesting a data transfer. Depending on the library routine called, it may be necessary for both sender and receiver to be "on the line" at the same time (which means that one will probably have to wait for the other to show up), or it is possible for the sender to send the message to a buffer, for later delivery, allowing the sender to proceed immediately to further computation.

### 2. Definition

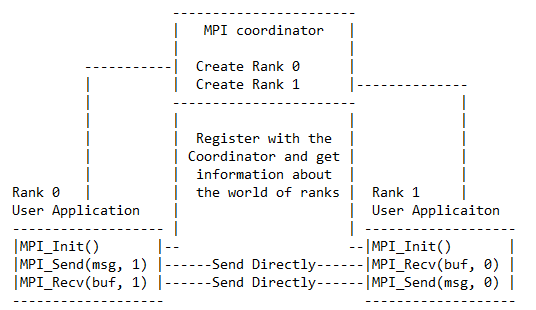
MPI is an API for passing messages among processes running on a cluster of servers. MPI is higher level than both TCP/IP and sockets. It can theoretically use any family of networking protocols, and if it's using TCP/IP or some other family that's supported by the sockets API, then it probably uses the sockets API to communicate with the operating system.

MPI is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementers, and users.

### 3. Advantages of MPI

* The MPI standard is available.
* MPI was designed for high performance on both massively parallel machines and on workstation clusters.
* MPI is widely available, with both free available and vendor-supplied implementations.
* MPI was developed by a broadly based committee of vendors, implementers, and users.
* Information for implementors of MPI is available.

### 4. Mechanism



* **Rank**: a unique integer process identifier.
* **Communicator**: a group of processes.

To pass messages between processes, each process must have a unique identifier. In the ubiquitous MPI standard, the identifier is known as the process rank. The rank space is an ordered set of nonnegative integers, enabling algorithms to communicate in a particular order.

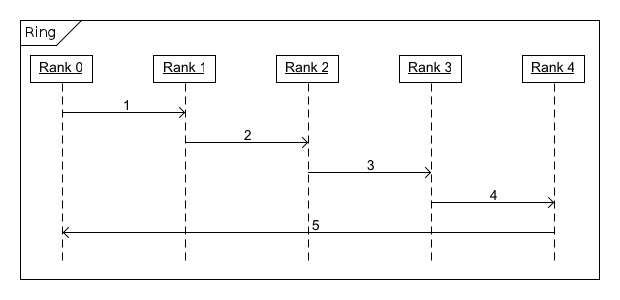


Figure 1 Diagram of rank MPI

Communication groups, called communicators, can be defined in order to partition the possible destinations that processes can send to. Only processes that belong to a communicator can send to other processes in that communicator. For the library and application not to receive each other’s MPI messages, one can separate them out into separate communicators, ensuring that messages sent by the library will never be received by the application, and vice versa. The hierarchical structure of this arrangement controls how the two communicators pass messages between themselves, which allows comprehensible communication between MPI jobs.

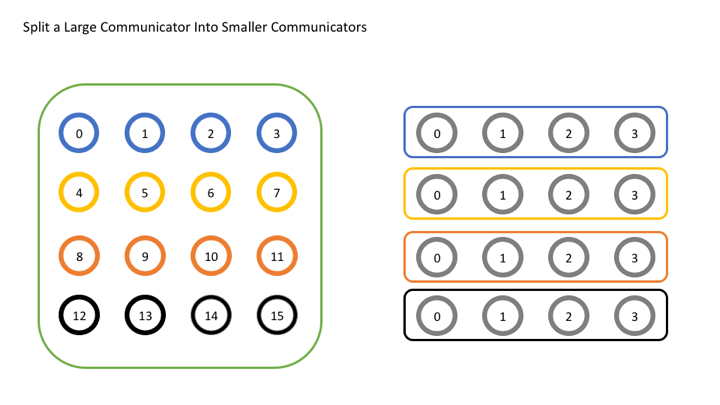


Figure 2 Diagram of MPI Communicator

In some cases the need for a defined communicator is unnecessary. MPI primarily addresses the message-passing parallel programming model: data is moved from the address space of one process to that of another process through cooperative operations on each process. MPI point-to-point operations typically involve message passing between two, and only two, different MPI tasks. One task is performing a send operation and the other task is performing a matching receive operation.

### 5. Noticeable function

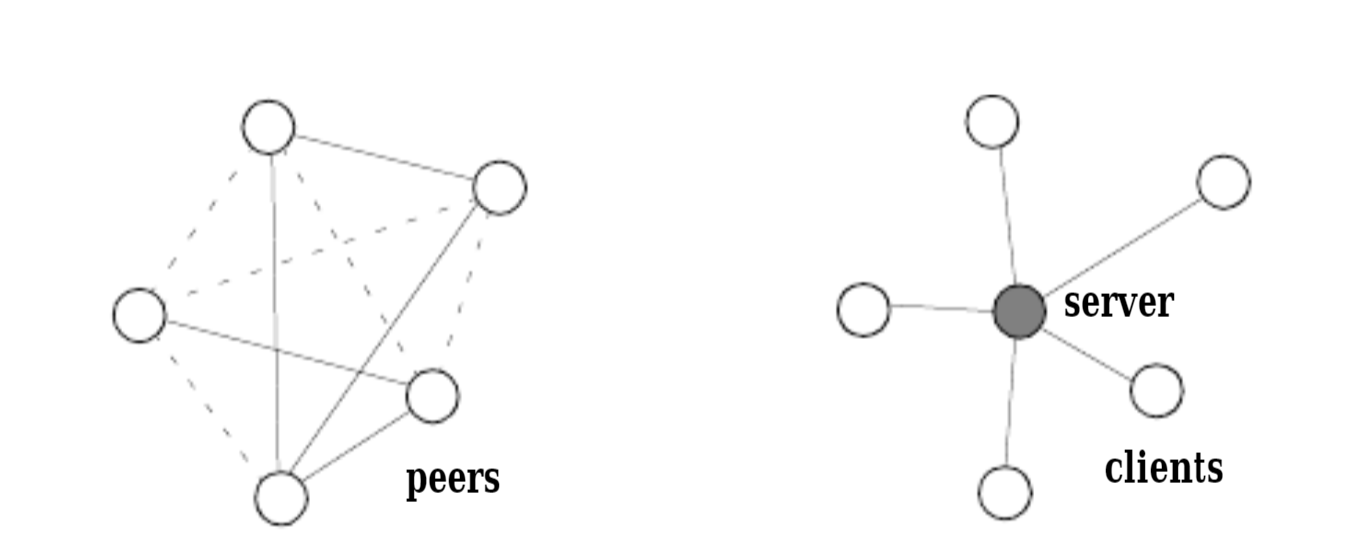
* MPI\_Send, to send a message to another process.
* int MPI\_Send(void \*data\_to\_send, int send\_count, MPI\_Datatype send\_type,  int destination\_ID, int tag, MPI\_Comm comm);
* data\_to\_send: messages
* send\_count: number of data elements to be sent
* send\_type: datatype of the data
* destination ID: process ID of destination (the rank of the receiving process)
* tag: message tag
* comm: communicator (handle)
* MPI\_Recv, to receive a message from another process.
* MPI\_Recv(void \*received\_data, int receive\_count, MPI\_Datatype receive\_type, int sender\_ID, int tag, MPI\_Comm comm, MPI\_Status \*status);
* received\_data: messages
* receive\_count: number of data elements expected
* receive\_type: datatype of the data to be received (one of the MPI datatype handles)
* sender\_ID: process ID of the sending process (the rank of sending process)
* tag: message tag
* comm: communicator (handle)
* status: status struct (MPI\_Status)
* MPI\_Init, initializes the MPI environment
* Must be called in every MPI program once in an MPI program, before any other MPI functions do.
* MPI\_Comm\_size, to obtain the count of processes
* MPI\_Comm\_size(comm, &process\_count);
* comm: communicator (handle)
* process\_count: count the number of process
* MPI\_Comm\_rank, obtain the rank of a process respectively [0 -> size - 1]
* MPI\_Comm\_rank(comm, &rank);
* comm: communicator (handle)
* rank: the rank of the process
* This function returns the rank of the processes respectively. For example:

process 1 -> rank 0, process 2 -> rank 1, ...

* MPI\_Abort, abort all MPI processes in a Communicator (MPI\_Comm)
* int MPI\_Abort(MPI\_Comm comm, int errorcode);
* comm: communicator (handle)
* errorcode: errorcode
* MPI\_Finalize, cleans up the MPI environment and ends MPI communications
* MPI\_Finalize();

# Peer-to-peer System (P2P)

### 1. Definition & Key value



In a “peer-to-peer” (P2P) system, each equally ranked autonomous peers depend on other alike peers (peers are autonomous when they are not wholly controlled by each other or by the same authority). Peers depend on each other for getting information, computing resources, forwarding requests, etc. As a result of the autonomy of peers, they cannot necessarily trust each other and rely completely on the behavior of other peers.

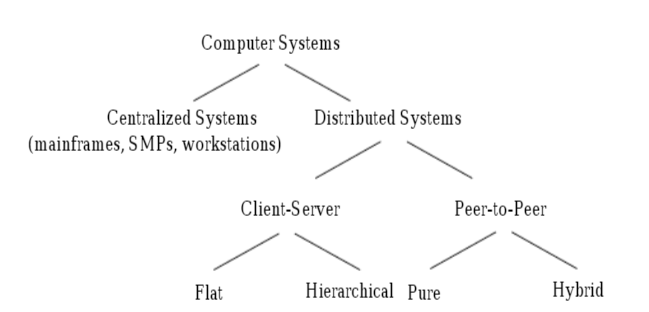
P2P refers to a class of system and application which employ distributed resources to perform a function in a decentralized manner. This also helps us improve scalability by avoiding dependency on centralized points and enabling resource aggregation.

In P2P, distributed computing nodes of equal roles or capabilities exchange information and services directly with each other. Each peer gives some resources and obtains other resources in return.

The critical function of P2P can be either distributed computing, data/content sharing, communication and collaboration, or platform services.

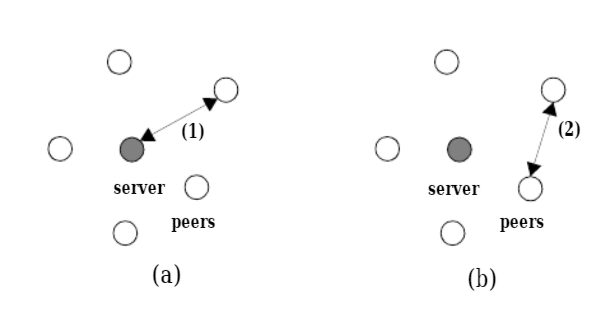
=> *That’s why we say: “****P2P is about sharing****”.*

### 2. Some terminologies



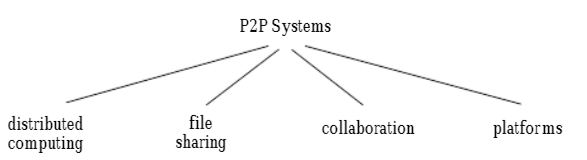
* **Centralized systems** represent single-unit solutions, including single and multi-processor machines, as well as high-end machines, such as supercomputers and mainframes.
* **Distributed systems** are those in which components located at networked computers communicate and coordinate their actions only by passing messages.
* **Client** is informally defined as an entity that initiates requests but is not able to serve requests. If the client also serves the request, then it plays the role of a server.
* **Server** is informally defined as an entity that serves requests from other entities, but does not initiate requests. If the server does initiate requests, then it plays the role of a client. Typically, there are one or a few servers versus many clients.
* **Client-Server model** represents the execution of entities with the roles of clients and servers. Any entity in a system can play both roles but for a different purpose. Similarly an entity can be a server for one kind of request and client for others.
* **Peer** is informally defined as an entity with capabilities similar to other entities in the system.
* **P2P model** enables peers to share their resources with at most a limited interaction with a centralized server. The peers may have to handle a limited connectivity, support possibly independent naming, and be able to share the role of the server. It is equivalent to having all entities being clients and servers for the same purpose.
* **Distributed computing**, which is defined as “a computer system in which several interconnected computers share the computing tasks assigned to the system”. Such systems include computing clusters, and global computing systems gathering computing resources from individual PCs over the Internet.

### 3. Taxonomy



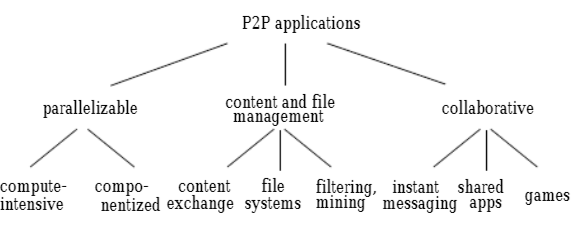
All computer systems can be classified into centralized and distributed. Distributed systems can be further classified into the client-server model and the P2P model. The client-server model can be flat where all clients only communicate with a single server, or it can be hierarchical for improved scalability. In a hierarchical model, the servers of one level are acting as clients to higher level servers. The P2P model can either be pure or it can be hybrid. In a pure model, there does not exist a centralized server.

### 4. P2P System



=> *Certain P2P systems emphasize different aspects along the taxonomy dimensions (computing, storage, communication), whereas the platforms support all of these dimensions.*

### 5. P2P Application



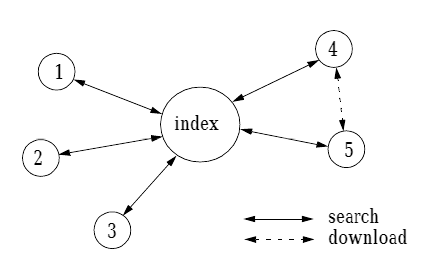
* **Parallel computing.** Parallelizable P2P applications split a large compute-intensive task into smaller sub pieces that can execute in parallel over a number of independent peer nodes. Most often, the same task is performed on each peer using different sets of parameters. envision applications that can be built out of finergrain components that execute over many nodes in parallel. In contrast to compute-intensive applications that run the same task on many peers, componentized applications run different components on each peer.
* **Content and file management.** Content and file management P2P applications focus on storing information on and retrieving information from various peers in the network.
* **Collaborative.** Collaborative P2P applications allow users to collaborate in real time, without relying on a central server to collect and relay information. Shared applications that allow people to interact while viewing and editing the same information simultaneously.

*=> Target environments for P2P: Consist of the Internet, intranets, and ad-hoc networks. P2P systems connected to Internet support connections in the spectrum from dialup lines to broadband (DSL). The underlying architecture can rely on personal home computers, corporate desktops, or personal mobile computers (laptops and handhelds).*

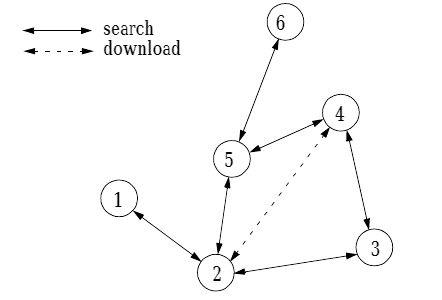
=> Infrastructure component of P2P should be covering:

* Communication paradigms.
* Peer group management.
* Security, resource aggregation, and reliability.
* Class specific.
* Application specific.

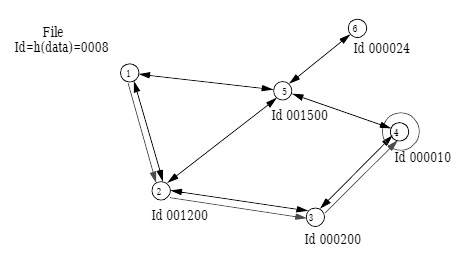
### 6. P2P Models and Algorithm



**Centralized directory model**. The peers of the community connect to a central directory where they publish information about the content they offer for sharing. Upon request from a peer, the central index will match the request with the best peer in its directory that matches the request. Then a file exchange will occur directly between the two peers. This model requires some managed infrastructure. This can cause the model to show some scalability limits.



**Flooded requests model.** This is a pure P2P model in which no advertisement of shared resources occurs. Instead, each request from a peer is flooded (broadcast) to directly connected peers until the request is answered or a maximum number of flooding steps (typically 5 to 9) occur. If the goal is to reach all the peers in the network, then this model does not prove to be very scalable, but it is efficient in limited communities such as a company network. By manipulating the number of connections of each node and appropriately setting the transistor-transistor-logic parameter of the request messages, the flooded requests model can scale to a reachable horizon of hundreds of thousands of nodes.



**Document routing model.**

->Each peer from the network is assigned a random ID and each peer also knows a given number of peers.

*When a document is published (shared) on such a system,*

->An ID is assigned to the document based on a hash of the document’s contents and its name.

->Each peer will then route the document towards the peer with the ID that is most similar to the document ID.

*This process is repeated until the nearest peer ID is the current peer’s ID.* *Each routing operation also ensures that a local copy of the document is kept.*

*When a peer requests the document from the P2P system,*

->The request will go to the peer with the ID most similar to the document ID.

*This process is repeated until a copy of the document is found.*

->Then the document is transferred back to the request originator, while each peer participating the routing will keep a local copy.

*Although the document routing model is very efficient for large, global communities, it has the problem that the document IDs must be known before posting a request for a given document. Hence it is more difficult to implement a search than in the flooded requests model.*

*=> Goals of each algorithm: to reduce the number of P2P hops to locate a document of interest and to reduce the amount of routing state that must be kept at each peer.*

### 8. Characteristic

One of the P2P’s characteristic is **Decentralization** but since our demo uses Hybrid Centralized and P2P, we may as well make the comparison as below:

* Centralized - Decentralized System comparison:

|  |  |  |
| --- | --- | --- |
|  | **Centralized** | **Decentralized** |
| *Illustration* |  |  |
| *Definition* | - Use client/server architecture where one or more client nodes are directly connected to a central server.  - Most commonly used. | - Every node in the system makes its own decision.  - Behavior: the aggregate of the decisions of the individual nodes.  - No single entity that receives and responds to the request. |
| *Characteristics* | - **Presence of a global clock:** As the entire system consists of a central node (a server/ a master) and many client nodes(a computer/ a slave), all client nodes sync up with the global clock(of the central node).  - **One single central unit:** One single central unit which serves/coordinates all the other nodes in the system.  - **Dependent failure of components:** Central node failure causes the entire system to fail. Because when the server is down, no other entity is there to send/receive response/requests. | - **Lack of a global clock:** Every node is independent of each other and hence, have different clocks that they run and follow.  - **Multiple central units** (Computers/Nodes/Servers): More than one central unit which can listen for connections from other nodes  - **Dependent failure of components:** One central node failure causes a part of system to fail; not the whole system. |
| *Components* | - Node.  - Server.  - Communication link. | - Node.  - Communication link. |
| *Architecture* | - **Client-Server architecture.** The central node that serves the other nodes in the system is the server node and all the other nodes are the client nodes. | - **P2P architecture.** All nodes are peers of each other. No one node has supremacy over other nodes  - **Master-slave architecture.** One node can become a master by voting and help in coordinating of a part of the system but this does not mean the node has supremacy over the other node which it is coordinating |
| *Limitations* | - **Can’t scale up vertically after a certain limit**. After a limit, even if you increase the hardware and software capabilities of the server node, the performance will not increase appreciably leading to a cost/benefit ratio < 1.   * - **Bottlenecks** can appear when the traffic spikes – as the server can only have a finite number of open ports to which can listen to connections from client nodes. So, when high traffic occurs like a shopping sale, the server can essentially suffer a [Denial-of-Service](https://www.geeksforgeeks.org/deniel-service-prevention/) attack or Distributed Denial-of-Service attack. | * May lead to **problem of coordination at the enterprise level**. When every node is owner of its own behavior, its difficult to achieve collective tasks * Not suitable for small systems * - **Not beneficial to build and operate small decentralized systems** because of low cost/benefit ratio * - **No way to regulate a node on the system**. No superior node overseeing the behavior of subordinate nodes |
| *Advantages* | * - **Easy to physically secure.** It is easy to secure and service the server and client nodes by virtue of their location * - **Smooth and elegant personal experience.** A client has a dedicated system which he uses(for example, a personal computer) and the company has a similar system which can be modified to suit custom needs * - **Dedicated resources** (memory, CPU cores, etc). * - **More cost efficient for small systems upto a certain limit.** As the central systems take less funds to set up, they have an edge when small systems have to be built * - **Quick updates are possible.** Only one machine to update. * - **Easy detachment of a node from the system.** Just remove the connection of the client node from the server and then the node is detached. | * - **Minimal problem of performance bottlenecks occurring.** The entire load gets balanced on all the nodes; leading to minimal to no bottleneck situations * - **High availability.** Some nodes(computers, mobiles, servers) are always available/online for work, leading to high availability * - **More autonomy and control over resources.** As each node controls its own behavior, it has better autonomy leading to more control over resources. |
| *Disadvantages* | - **Highly dependent on the network connectivity.** System can fail if the nodes lose connectivity as there is only one central node.   * - **No graceful degradation of system.** Abrupt failure of the entire system * Less possibility of data backup. If the server node fails and there is no backup, you lose the data straight away * - **Difficult server maintenance.** There is only one server node and due to availability reasons, it is inefficient and unprofessional to take the server down for maintenance. So, updates have to be done on-the-fly (hot updates) which is difficult and the system could break. | * - **Difficult to achieve global big tasks.** No chain of command to command others to perform certain tasks * - **No regulatory oversight.** * - **Difficult to know which node failed.** Each node must be pinged for availability checking and partitioning of work has to be done to actually find out which node failed by checking the expected output with what the node generated * - **Difficult to know which node responded.** When a request is served by a decentralised system, the request is actually served by one of the nodes in the syste |
| *Application* | * - **Application development.** Very easy to setup a central server and send client requests. Modern technology these days do come with default test servers which can be launched with a couple commands. For example, express server, django server. * - **Data analysis.** Easy to do data analysis when all the data is in one place and available for analysis * - **Personal computing.** | * - **Private networks.** Peer nodes joined with each other to make a private network. * - **Cryptocurrency.** Nodes joined to become a part of a system in which digital currency is exchanged without any trace and location of who sent what to whom. |

* Other characteristics of P2P:
* **Scalability:**

An immediate benefit of decentralization is improved scalability. Scalability is limited by factors such as the amount of centralized operations that needs to be performed, the amount of state that needs to be maintained, the inherent parallelism an application exhibits, and the programming model that is used to represent the computation.

Scalability also depends on the ratio of communication to computation between the nodes in a P2P system. Applications, such as code breaking or prime number search have the ratio close to zero, making it extremely scalable.

Recent P2P systems dictate a consistent mapping between an object key and hosting node. Therefore, an object can always be retrieved as long as the hosting nodes can be reached. Nodes in these systems compose an overlay network. Each node only maintains information about a small number of other nodes in the system. This limits the amount of state that needs to be maintained, and hence increases scalability.

* **Anonymity:**

One goal of P2P is to allow people to use systems without concern for legal or other ramifications. Another goal is to guarantee that censorship of digital content is not possible.

* **Author:** A document’s author or creator cannot be identified
* **Publisher:** The person who published the document to the system cannot be identified
* **Reader:** People who read or otherwise consume data cannot be identified
* **Server:** Servers containing a document cannot be identified based on the document
* **Document:** Servers do not know what documents they are storing
* **Query:** A server cannot tell what document it is using to respond to a user’s query

Regardless of the above entity, there are three different kinds of anonymity between each communicating pair: sender anonymity, which hides the sender’s identity; receiver anonymity, which hides a receiver’s identity; and mutual anonymity, which hides the identities of the sender and receiver are hidden from each other and other peers:

* **Self-organization:**

Self-organization is defined as “a process where the organization (constraint, redundancy) of a system spontaneously increases, i.e., without this increase being controlled by the environment or an encompassing or otherwise external system”

In P2P systems, self-organization is needed because of scalability, fault resilience, intermittent connection of resources, and the cost of ownership. P2P systems can scale unpredictably in terms of the number of systems, number of users, and the load. It is very hard to predict any one of them, requiring frequent re-configuration of centralized systems. The significant level of scale results in an increased probability of failures, which requires self-maintenance and self-repair of the systems. Similar reasoning applies to intermittent disconnection; it is hard for any predefined configuration to remain intact over a long period of time. Adaptation is required to handle the changes caused by peers connecting and disconnecting from the P2P systems. Finally, because it would be costly to have dedicated equipment and/or people for managing such a fluctuating environment, the management is distributed among the peers.

* **Performance:**

Performance is a significant concern in P2P systems. P2P systems aim to improve performance by aggregating distributed storage capacity and computing cycles of devices spread across a network. Because of the decentralized nature of these models, performance is influenced by three types of resources: processing, storage, and networking. In particular, networking delays can be significant in wide-area networks. Bandwidth is a major factor when a large number of messages are propagated in the network and large amounts of files are being transferred among many peers. This limits the scalability of the system.

*=> Keys to approaches optimization in performance:*

* **Replication.** Replication puts copies of objects/files closer to the requesting peers, thus minimizing the connection distance between the peers requesting and providing the objects. Changes to data objects have to be propagated to all the object replicas. sIn combination with intelligent routing, replication helps to minimize the distance delay by sending requests to closely located peers. Replication also helps to cope with the disappearance of peers. Because peers tend to be user machines rather than dedicated servers, there is no guarantee that the peers won't be disconnected from the network arbitrarily.
* **Caching.** Caching reduces the path length required to fetch a file/object and therefore the number of messages exchanged between the peers. Reducing such transmissions is important because the communication latency between the peers is a serious performance bottleneck facing P2P systems. In Freenet for example, when a file is found and propagated to the requesting node,the file is cached locally in all the nodes in the return path. More efficient caching strategies can be used to cache large amounts of data infrequently. The goal of caching is to minimize peer access latencies, to maximize query throughput and to balance the workload in the system. The object replicas can be used for load balancing and latency reduction.
* **Intelligent routing and network organization.** To fully realize the potential of P2P networks, it is important to understand and explore the social interactions between the peers. Establishing a good set of peers reduces the number of messages broadcast in the network and the number of peers that process a request before a result is found. By proactively moving the data in the network, the peers decide whom to contact and when to add/drop a connection based on local information only.

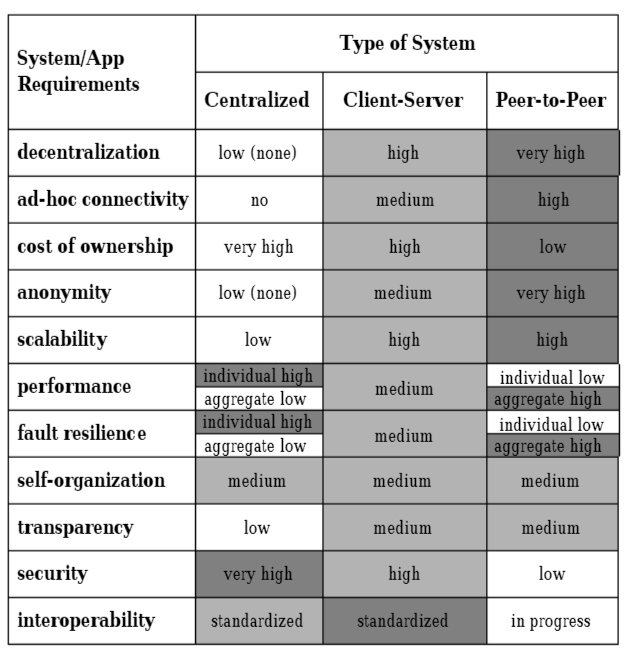
### 9. Security

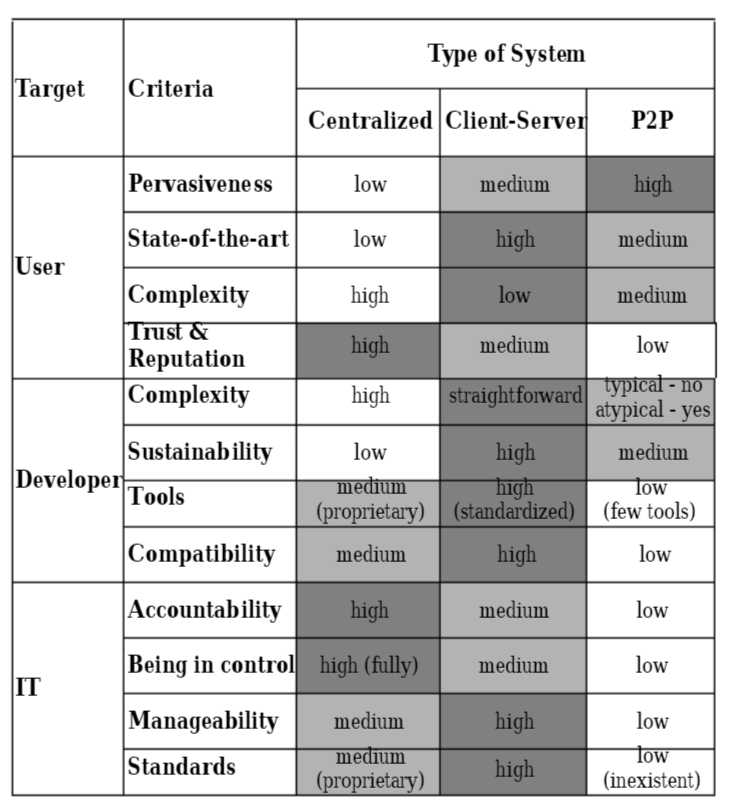
P2P systems share most of their security needs with common distributed systems: trust chains between peers and shared objects, session key exchange schemes, encryption, digital digests, and signatures. Security requirements appeared with P2P systems as following:

* **Multi-key encryption.** File sharing systems intended to protect a shared object, as well as the anonymity of its author, publishing peer and hosting peer.
* **Sandboxing.** Some distributed computing P2P systems require downloading codeon peer machines. It is crucial to protect the peer machines from potentially malicious code and protect the code from a malicious peer machine. Protecting a peer machine typically involves enforcing:
  + Safety properties such that the external code will not crash the host box, or will only access the host data in a type-safe way.
  + Security properties to prevent sensitive data from being leaked to malicious parties.
* **Digital Rights Management.** P2P file sharing makes file copied easy. It is necessary to be able to protect the authors from having their intellectual property stolen. One way to handle this problem is to add a signature in the file that makes it recognizable (the signature remains attached to the file contents) although the file contents do not seem affected.
* **Reputation and Accountability.** In P2P systems, it is often important to keep track of the reputation of peers to prevent ill-behaved peers from harming the whole system. Reputation requires ways to measure how “good” or “useful” a peer is. For instance, if a given user shares lots of interesting files, its reputation should be high. Freeloader is a common term for users who download files from P2P systems without offering files to others. A freeloader usually has a low reputation. To prevent this kind of non-cooperative behavior, some accountability mechanisms need to be devised
* **Firewalls.** P2P applications inherently require direct connections between peers. However, in corporate environments internal networks get isolated from the external network (the Internet), leaving reduced access rights to applications.

### 10. Strengths and Weaknesses

The big advantage of P2P is the resources of many users and computers can be brought together to yield large pools of information and significant computing power. Furthermore, because computers communicate directly with their peers, network bandwidth is better utilized. However, there are often inherent drawbacks to P2P solutions precisely because of their decentralized nature.





# Hybrid Centralization and P2P System

### 1. Hybrid Procurement Systems

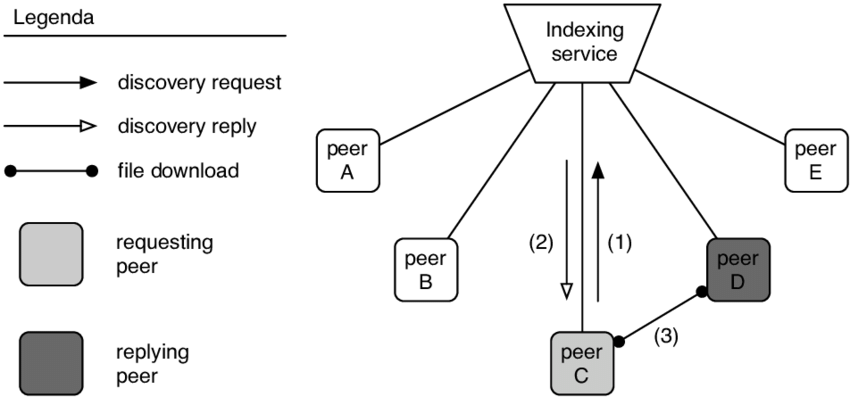
Hybrid models adopt a pick-and-choose approach to procurement, with a central leadership structure supported by local managers with some degree of autonomy.

Advances in technology have made this easier, as communication and working processes can be more nimble, letting directives filter through hybridized structures more quickly than they could even a few years ago.

The most effective hybridized systems maximize communication along the organizational chart to ensure core standards are maintained and aggressively take advantage of technology to facilitate those goals.

Hybrid models represent an attempt to reap the strongest benefits of both centralized and decentralized procurement. There are three main hybrid alternatives

### 2. Hybrid P2P



The hybrid P2P is regarded as centralized, but not in the sense of a general-purpose server. It has router-terminasls that are used for holding a catalog of addresses. The addresses are referenced by a set of indexes, which determines the appropriate address set. This router-terminal could be dynamic or static.

If it is dynamic then the router-terminal could be characterized as a peer group manager. However, we could diminish the functions of the centralized terminal until it is nothing more than a router database controlled by the peer interface.

As a router database, all powers of management remain with the peers. This eliminates the centralized computer (router-terminal set) as a peer group manager. All management functions are then held in the peers.

All the peers in a peer group have equal and similar status. The more management power we take out of a peer (and therefore out of all of them since all peers have equal management powers) the more we degrade the P2P architecture.

This implies that even if the routers can dynamically update their routing catalogs (router-terminals functioning as agents), that the use of these catalogs is within the management scope of every peer.

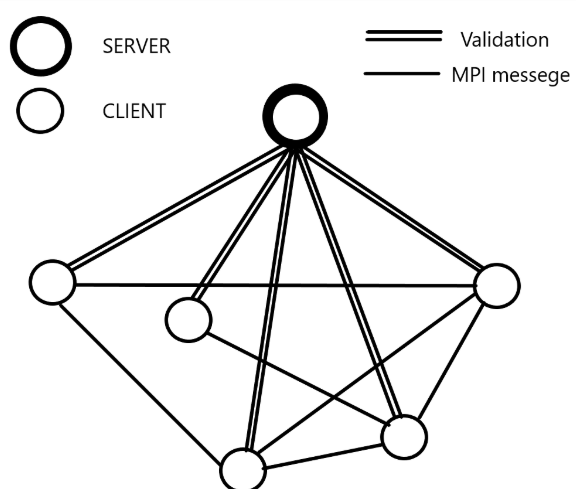
It is feasible that other than dynamic updates done by the router-terminals (possible agent-driven) that each peer could have the power for updating the database, at least in regard to what they are willing to store and upload.

Router-terminals, as catalogs of addresses, are referenced by indexes. Independent of upgrades, there are two general ways for hybrid P2P to utilize the router terminals.

### 3. Requirements

* Must be easy to comprehend.
* Suitable for both small and large scale for educational and future improvement as well as implementation.
* The system must maximize the advantages of each kind of system while minimize their disadvantages.
* There must be a hierarchy to oversight and enforce rule over the client nodes.
* Client node can interact directly with each other in the network after validification with the server.

### 4. Implementation



Hybrid-centralized system is a hierarchy system where the server node work is only to oversight and validate client nodes, each client node has the freedom to connect directly to others nodes. After first validification the node can still communicate with each other even when the server is offline.

# Load Testing

The popular load testing tools available also provide insight into the causes for slow performance. There are numerous possible causes for slow system performance, including, but not limited to, the following:

* Application server(s) or software
* Database server(s)
* Network – latency, congestion, etc.
* Client-side processing
* Load balancing between multiple servers

Load testing is especially important if the application, system or service will be subject to a [service level agreement](https://en.wikipedia.org/wiki/Service_level_agreement) or SLA.

### 1. User experience under load test

### **Stress testing**

* To determine breaking points or safe usage limits.
* To confirm mathematical model is accurate enough in predicting breaking points or safe usage limits.
* To confirm intended specifications are being met.
* To determine modes of failure.
* To test stable operation of a part or system outside standard usage.

*Reliability engineers often test items under expected stress or even under accelerated stress in order to determine the operating life of the item or to determine modes of failure.*

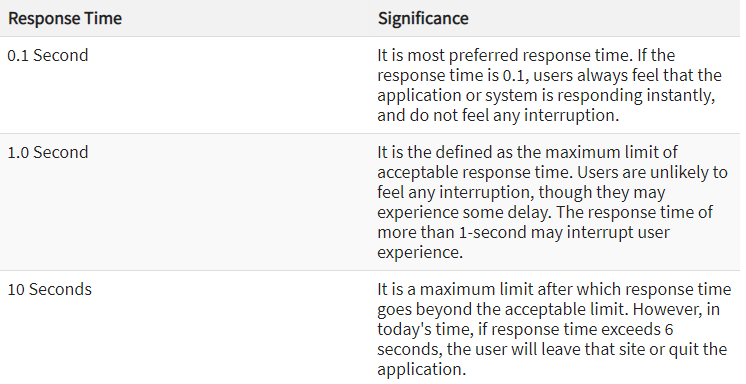
### 2. Three important response time values:

Response time testing has two most essential characteristic:

* Average response time
* Maximum response time.

*It shows how long a user needs to wait for the server to the response its request*.

**Key response time values**



### 3. Creating a Benchmark Test Plan

Test plan is the most crucial element of the benchmark testing process. Benchmark Test plan follows steps as given below;

1. *Scaling and invoking workload*
2. *Collect and store measures for benchmark testing*
3. *Define the time span required and the terminal point of a test process*
4. *Prepare a backup plan to overcome any new test case failure*
5. *Decide the authority to call termination of the end process*

* Load/Stress Test Types and Schedules The following tests will be run:
* **Capacity Test**: Determines the maximum number of concurrent users that the application server can support under a given configuration while maintaining an acceptable response time and error rate
* **Consistent Load Test:** Long-running stress test that drives a continuous load on the application server for an extended period of time (at least 6 hours). The main purpose of this type of test is to ensure the application can sustain acceptable levels of performance over an extended period of time without exhibiting degradation, such as might be caused by a memory leak.
* **Single Function Stress Test:** A test where 100 users perform the same function with no wait times and no ramp up time. This test will help determine how the application reacts to periods of extreme test in a very narrow area of the code.
* **Baseline Test:** At the conclusion of the Capacity Test and Consistent Load Test a third test will be established with the goal to be a repeatable test that can be performed when any portion of the system is changed. This test will not have the secondary goals

*The average response time of a process is usually around 0.4s - 0.6s*

\* Scalability is limited by factors such as:

* The amount of centralized operations that needs to be performed
* The amount of state that needs to be maintained
* The inherent parallelism an application exhibits,
* The programming model that is used to represent the computation.
* The ratio of communication to computation between the nodes in a P2P system.

### 4. Why should we use MPI

|  |  |  |
| --- | --- | --- |
| **API Type** | **Socket** | **MPI** |
| **Impact** | You can use any family of networking protocols through the sockets API, the protocol family is a parameter that you pass in. | You can use any family of networking protocols, and if it's using TCP/IP or some other family that's supported by the sockets API. |
| *⇒ MPI is higher level than sockets.*  *MPI might use sockets.* | | |

*Most MPI implementations use sockets for TCP based communication. Odds are good that any given MPI implementation will be better optimized and provide faster message passing, than a home grown application using sockets directly.*

# Further Details

## Members and Duties

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Name** | **ID** | **Research** | **Practical Work** |
| **1** | Vũ Đức Hiếu | BI9 – 102 | **LEADER  Coding Preparation** | **Code** |
| **2** | Trần Trường Giang | BI9 – 090 | **PRESENTER**  **Centralized & Decentralized System** | **Code** |
| **3** | Đào Minh Vũ | BI9 – 237 | **RESEACHER**  **Message Passing Interface (MPI)** | **Code** |
| **4** | Trần Đại Dương | BI9 – 079 | **REPORTERS**  **Applying Hybrid Centralization and Peer-to-peer (P2P)** | **Slide** |
| **5** | Đặng Yến Nhi | BI9 – 180 | **Report** |

## Behind the scenes

#### [This link mostly shows our working process and contains all the references included in our report.](https://docs.google.com/spreadsheets/d/1gywPoP5-WmyK-mFRZ6q9_Zm5vYSHTlAEcEO6LOhTV1w/edit?usp=sharing)