1. What is the role of the middleware in a distributed system?

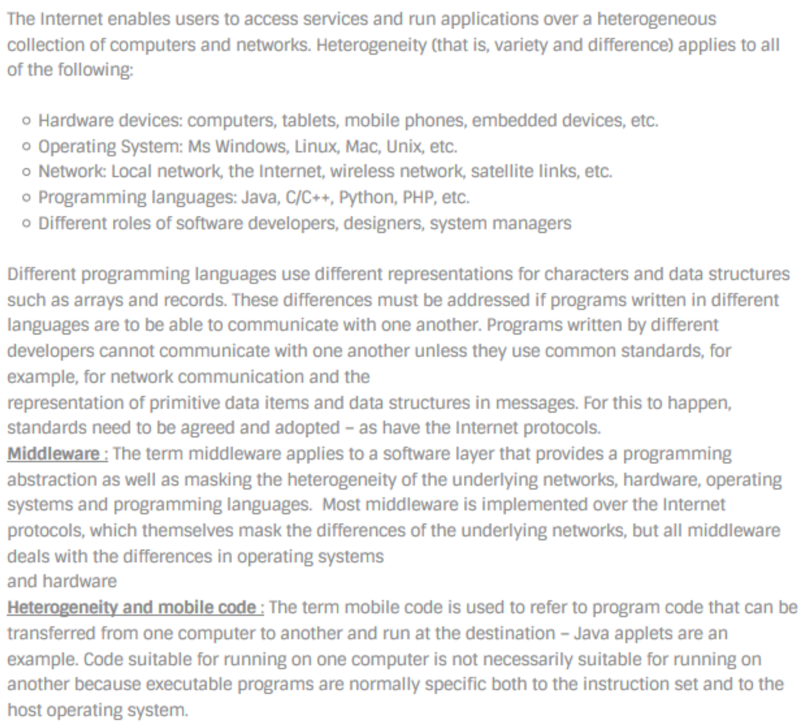
   To assist the development of distributed applications, distributed systems are often organised to have a separate layer of software that is logically placed on top of the respective operating systems of the computers that are part of the system. In a sense, middleware is the same to a distributed system as what an operating system is to a computer: a manager of resources offering its applications to efficiently share and deploy those resources across a network. Next to resource management, it offers services that can also be found in most operating systems, including:

   - Facilities for interapplication communication
   - Security services
   - Accounting services
   - Masking if and recovery from failures

   The main difference with their operating-system equivalents, is that middleware services are offered in a networked environment. Note also that most services are useful to many applications. In this sense, middleware can also be viewed as a container of commonly used components and functions that now no longer have to be implemented by applications separately.

2. Give one example of heterogeneity in a distributed system. Why is heterogeneity a challenge in distributed systems?

   Example is client server environment. Middelware is used to enable the application and end-users to interact with each other.
   Heterogeneity refers to the ability for the system to operate on a variety of different hardware and software components. This is achieved through the implementation of middle-ware in the software layer. The goal of the middle-ware is to abstract and interpret the programming procedural calls such that the distributed processing can be achieved on a variety of differing nodes.


The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. Heterogeneity (that is, variety and difference) applies to all of the following:

- Hardware devices: computers, tablets, mobile phones, embedded devices, etc.
- Operating System: Ms Windows, Linux, Mac, Unix, etc.
- Network: Local network, the Internet, wireless network, satellite links, etc.
- Programming languages: Java, C/C++, Python, PHP, etc.
- Different roles of software developers, designers, system managers

Different programming languages use different representations for characters and data structures such as arrays and records. These differences must be addressed if programs written in different languages are to be able to communicate with one another. Programs written by different developers cannot communicate with one another unless they use common standards, for example, for network communication and the
representation of primitive data items and data structures in messages. For this to happen, standards need to be agreed and adopted – as have the Internet protocols.
Middleware : The term middleware applies to a software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages. Most middleware is implemented over the Internet protocols, which themselves mask the differences of the underlying networks, but all middleware deals with the differences in operating systems
and hardware
Heterogeneity and mobile code : The term mobile code is used to refer to program code that can be transferred from one computer to another and run at the destination – Java applets are an example. Code suitable for running on one computer is not necessarily suitable for running on another because executable programs are normally specific both to the instruction set and to the host operating system.

3. What advantages and disadvantages can you see with software-as-a-Service?

4. List the ACID properties?

In the context of transaction processing, the acronym *ACID* refers to the four key properties of a transaction: atomicity, consistency, isolation, and durability.

Atomicity

All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.
For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.

Consistency

Data is in a consistent state when a transaction starts and when it ends.
For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

Isolation

The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized.
For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

Durability

After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure.
For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.

5. Is a reduction a synchronous operation? Do not forget to motivate your answer

   Yes, Reduction a synchronous operation as synchronous operations require that one operation must wait for another to complete before it can begin. As an example, consider a grocery checkout line: before the customer can pay for their groceries the checkout clerk must finish scanning all of the groceries. Scanning and paying are two distinct operations, but the operation of paying cannot proceed until the operation of scanning the products is complete. This operation is fairly inefficient, since neither the buyer nor payer can do their part of the transaction until the other has completed. To service more customers, it can be tempting to think that the solution is to open more lanes, when in fact a better solution is to optimize what happens within each checkout lane.

6. Why are most parallel computer systems MIMD systems?

   MIMD is more expensive but can perform much more complex programs. Also, MIMD can multitask and perform multiple processes at the same time. MIMD is the most basic and most familiar type of parallel processor. MIMD architecture includes a set of N-individual processors. Each processor includes their own memory that can be common to all processors. Each processor can operate independently and asynchronously. Many of the processors may carry out various instructions at any time on various pieces of data. The two most prominent types of parallel computing both belong to MIMD architecture. These are known as the shared memory MIMD and distributed memory MIMD. Shared created a group of memory modules while distributed clones the memory/processor pairs. By providing every processor its own memory, the MIMD architecture bypasses the downsides of SIMD. A processor may only access the memory that is directly connected to it.

7. There are several ways in which we write programs that yield good temporal locality. Discuss two examples of common programming approaches that yield good temporal locality

   The outer loop is an example of spacial locality. It sequentially increments the address the inner for-loop calls.
   The inside loop demonstrates temporal locality. The exact same memory address is accessed ten times in a row, and multiplied by j each time.

Spatial and temporal locality describe two different characteristics of how programs access data (or instructions).

> A sequence of references is said to have spatial locality if things that are referenced close in time are also close in space (nearby memory addresses, nearby sectors on a disk, etc.). A sequence is said to have temporal locality if accesses to the same thing are clustered in time.

If a program accesses every element in a large array and reads it once and then moves on to the next element and does not repeat an access to any given location until it has touched every other location then it is a clear case of *spatial* locality but not *temporal* locality. On the other hand, if a program spends time repeatedly accessing a random subset of the locations on the array before moving on to another random subset it is said to have *temporal* locality but not *spatial* locality. A well written program will have data structures that group together things that are accessed together, thus ensuring *spatial* locality. If you program is likely to access **B** soon after it accesses **A** then both **A** and **B** should be allocated near each other.

Your first example

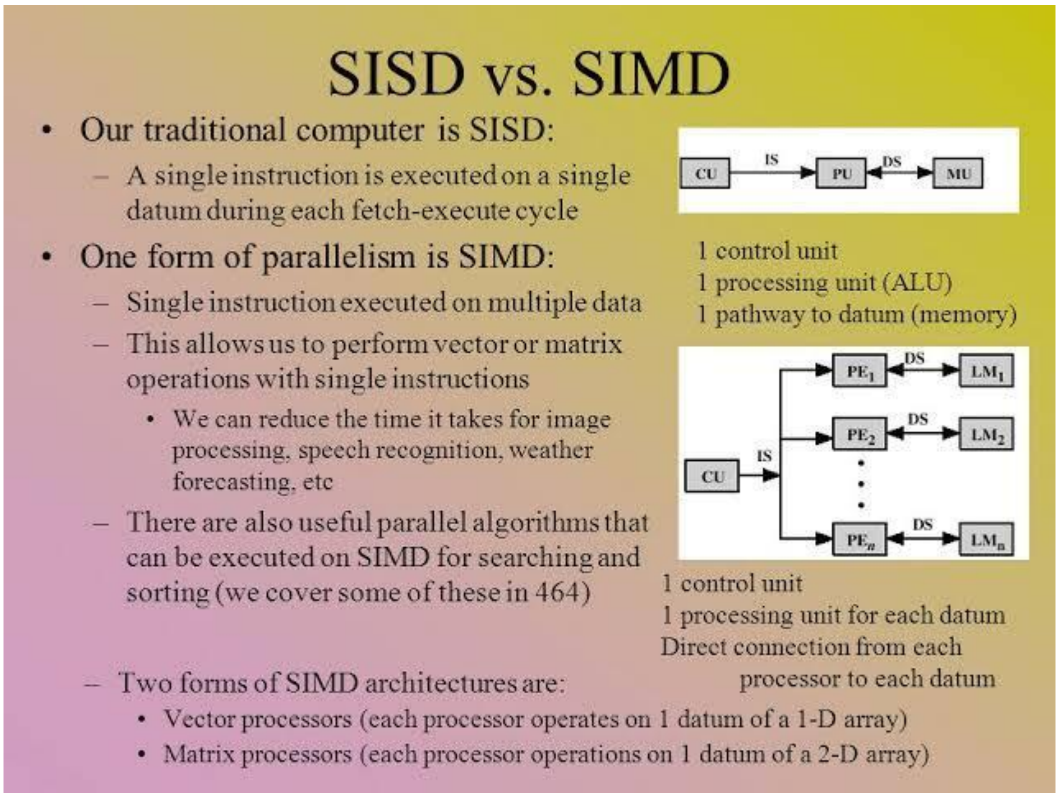
A[0][1], A[0][2], A[0][3]
shows *spatial* locality, things that are accessed close in time are close in space. It does not show *temporal* locality because you have not accessed the same thing more than once.

Your second example
A[1], A[2], A[3]
also shows *spatial* locality, but not *temporal* locality.

Here's an example that shows *temporal* locality

A[1], A[2000], A[1], A[1], A[2000], A[30], A[30], A[2000], A[30], A[2000], A[30],

8. Explain in your own words why SIMD may be advantageous over SISD.



SISD vs. SIMD

- Our traditional computer is SISD:
  - A single instruction is executed on a single datum during each fetch-execute cycle
- One form of parallelism is SIMD:
  - Single instruction executed on multiple data
  - This allows us to perform vector or matrix operations with single instructions
    - We can reduce the time it takes for image processing, speech recognition, weather forecasting, etc
  - There are also useful parallel algorithms that can be executed on SIMD for searching and sorting (we cover some of these in 464)
  - Two forms of SIMD architectures are:
    - Vector processors (each processor operates on 1 datum of a 1-D array)
    - Matrix processors (each processor operations on 1 datum of a 2-D array)

1 control unit
1 processing unit (ALU)
1 pathway to datum (memory)

1 control unit
1 processing unit for each datum
Direct connection from each processor to each datum

SISD means single instruction single data. It means that an instruction will be working on a given set of operands.

In such processors, the hardware is designed in such a way that it can hold operands for only one instruction at a time.

SIMD on the other hand has great advantage in situations where same instruction/operation is applied on multiple data again and again.

Eg. Adding two array using a for loop. Here the instruction in same, a[i] + b[i], but there would be several iterations based on the lengths of these arrays.

SIMD processors such as vector processors have special registers (vectors) where they can hold large number of data elements at once and process them in one go.

In this case, elements of arrays a and b will be held by two vectors and then addition will be carried out simultaneously on all the elements in both vectors.

Hence SIMD is more powerful, less time consuming but in particular situations only.

9. Edge Computing and Cloud Computing are established computing paradigms that can allow for faster response times and handling of the large amounts of data produced by Internet-of-Things, IoT, solutions.

What are the motivations for moving to a paradigm where code is run along the continuum across many computing platforms including the edge and the cloud?

There are several reasons why the edge computing market is increasing rapidly. In addition to the greater number of connected devices, there are three key drivers of growth for edge computing:

1. **Latency** — For many time-sensitive applications, the process being monitored requires a response in near real time, with near-zero latency. In these situations a round trip of data to and from the cloud or a corporate data center is impractical.
2. **Bandwidth** — Both the physical limits of available bandwidth and the cost of transmitting large quantities of data make edge computing an attractive alternative.
3. **Reliability** — Network congestion can interrupt the flow of data, causing unacceptable interruptions in use cases such as point-of-sale systems.

Examples

Autonomous Vehicles, Electric Vehicles and Charging Stations

To function safely, autonomous vehicles need to collect and process data about their location, direction, speed, traffic conditions and more — all in real time. This involves sufficient onboard computing capacity to make every autonomous vehicle, in effect, its own network edge. Edge computing devices can gather data from vehicle sensors and cameras, process it and make decisions in milliseconds, with virtually no latency.

Healthcare and Medical Applications

The healthcare and medical industries collects patient data from sensors, monitors, wearables and other equipment to provide healthcare professionals with accurate, timely insights on patient condition.

Edge computing solutions can deliver that information to dashboards for a complete, at-a-glance view of important indicators.

Edge computing solutions equipped with artificial intelligence (AI) and machine learning can identify outlier data so that medical professionals can respond to health needs in real time with a minimum of false alarms.

Cloud computing and edge computing are well known for their distinct advantages in IoT based on use case, data processing and storage needs. However, the combination of the two computing infrastructures offers greater flexibility to developers and lower latency to consumers while also maintaining data privacy standards.
Enter the concept of cloud at the edge, a term gaining traction among behemoth cloud service providers, network operators and IoT developers.

What does edge cloud mean?

To understand cloud at the edge, otherwise known as edge cloud, tech experts must also define the two terms it combines and the differences between them.

Cloud computing refers to the storage and processing of data in a community, private, public or hybrid cloud data center in a centralized location. For IoT applications, processing all data in the cloud introduces greater latency to complete an action.

Edge computing refers to the process of real-time data storage and computation on the device or data source, rather than sending it to a distant data center. For IoT devices, this dramatically decreases lag and saves bandwidth. The centralized cloud still serves as the main storage facility for large amounts of data and additional processing. The IoT device where edge processing occurs acts as a node.

Edge cloud refers to the decentralization of the traditional, massive cloud data centers. The IoT edge cloud moves cloud storage and compute closer to the edge source while also scaling down its size. Edge sites may connect to each other or to a core cloud for additional data inputs and processing or storage capabilities, or isolated in instances of a data breach or service compromise.

Edge cloud requires additional remotely administered data centers -- also referred to as edge sites -- close to the end users. It also calls for a large number of edge sites at specific locations where increased compute and processing is needed beyond what can be completed at the edge in conjunction with low-latency, time-sensitive IoT tasks.

considerations of edge cloud computing

Even though the tools and architectures needed to build out the cloud at the edge are still nascent, opportunities abound when considering future edge cloud computational capabilities as applied to distributed node edge sites. Organizations can use IoT edge cloud computing for machinery analytics processing, execution in industrial IoT and ultrafast telematics processing in metro areas with a high number of autonomous vehicles.

Developers and IT professionals should plan now for both opportunities presented and potential constraints. The IoT edge cloud offers consistent OS deployment and application integration among edge sites to increase regularity of service delivery. Organizations can also use edge clouds to scale globally and remotely with an increased distribution of cloud nodes where clusters of users will benefit most. IT experts must be aware of edge cloud network limitations in connectivity and reliability to consistently meet low-latency IoT application requirements, such as those in augmented or virtual reality. The IoT edge cloud also requires hardware consistency and can be cost-prohibitive if not fulfilled.

Additional areas where decreased latency, high data processing and increased cost-efficiency of edge cloud data center distribution can have sizable effects include:

clusters of users with high streaming of video on mobile devices;

within cities or near highways to administer public safety applications at scale;

manufacturers or logistics centers that utilize robotics, automation and data analytics; and

hospitals, clinics and other healthcare facilities that require immediacy in IoT service delivery.

In these situations, edge sites may even predict data processing demands based on historical data analysis and predictive analytics capabilities, connecting to other nodes on the edge cloud to respond to variable processing needs.

Opportunities, considerations of edge cloud computing

Even though the tools and architectures needed to build out the cloud at the edge are still nascent, opportunities abound when considering future edge cloud computational capabilities as applied to distributed node edge sites. Organizations can use IoT edge cloud computing for machinery analytics processing, execution in industrial IoT and ultrafast telematics processing in metro areas with a high number of autonomous vehicles.

Developers and IT professionals should plan now for both opportunities presented and potential constraints. The IoT edge cloud offers consistent OS deployment and application integration among edge sites to increase regularity of service delivery. Organizations can also use edge clouds to scale globally and remotely with an increased distribution of cloud nodes where clusters of users will benefit most. IT experts must be aware of edge cloud network limitations in connectivity and reliability to consistently meet low-latency IoT application requirements, such as those in augmented or virtual reality. The IoT edge cloud also requires hardware consistency and can be cost-prohibitive if not fulfilled.

Additional areas where decreased latency, high data processing and increased cost-efficiency of edge cloud data center distribution can have sizable effects include:

clusters of users with high streaming of video on mobile devices;

within cities or near highways to administer public safety applications at scale;

manufacturers or logistics centers that utilize robotics, automation and data analytics; and

hospitals, clinics and other healthcare facilities that require immediacy in IoT service delivery.

In these situations, edge sites may even predict data processing demands based on historical data analysis and predictive analytics capabilities, connecting to other nodes on the edge cloud to respond to variable processing needs.

IoT edge cloud success depends on laying out the groundwork

For the IoT edge cloud to reach its full potential, organizations must have technologies in place that solve not only intelligent edge processing needs, but also network infrastructure limitations.

AI chips within IoT devices process more on the edge directly. That, combined with a shorter distance to an edge site, equals faster replies or actions while also saving on bandwidth and server costs. For example, AI chips in devices provide smarter and quicker answers to Alexa queries or AI-driven photo editing.

Progressive web applications offer save-as-you-go changes stored locally, even if offline, that sync with the cloud when reconnected. Edge sites can use the applications to balance the workload between processors with more frequent and flexible synchronization.

5G network connectivity is critical to cloud edge sites operating and communicating at the speeds necessary to maintain low latency as compared to edge computing alone. For example, AWS Wavelength and Verizon 5G have unveiled products that developers can use to build apps directly at the edge with access to the density and capacity of a 5G network.

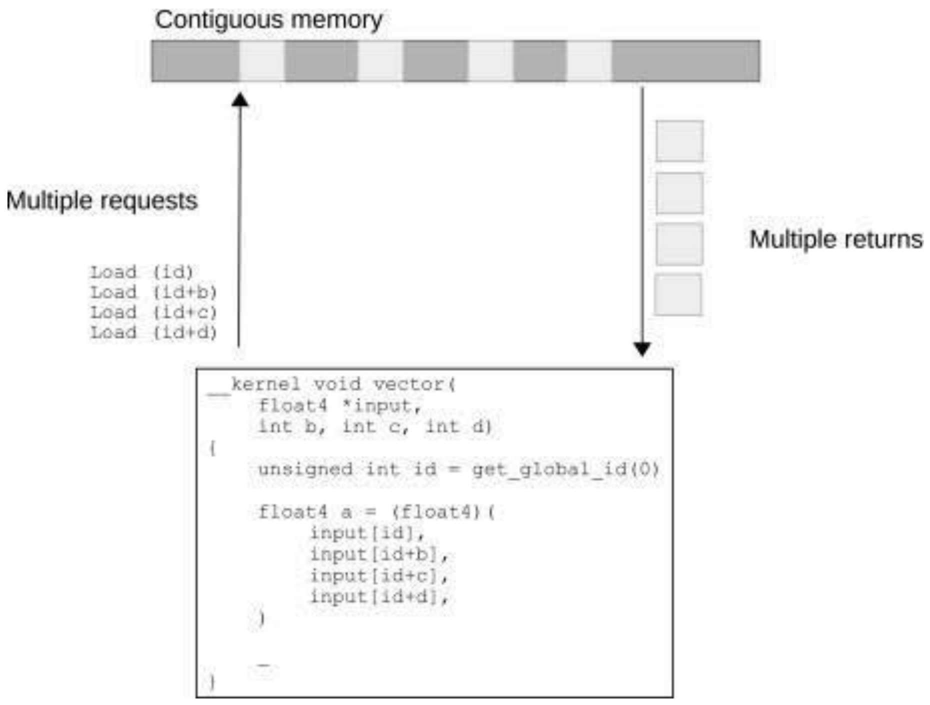
10. Explain in your own words, and with an example, how you can improve spatial locality of a program.

Improving of spatial locality of a program

1.1.2 Spatial/Temporal Locality

When discussing cache efficiency there are two very important concepts, temporal locality and spatial locality. These are the fundamental ideas behind how the cache operates and how it is built. Performance can be greatly increased if the programs are structured to use these localities efficiently. Temporal locality says that if one memory area is used it is likely that it will be used again soon. This is helped by the associativity of the cache, making the data stay for longer periods of time.

Spatial locality is if one memory area is used then an area close by is going to be used in the near future. How much area that should be prepared is always a bit of a guess as this depends on several factors like the average size of functions. Spatial locality can also be improved by increasing the likelihood that one of the areas around the called memory area contain useful data.



```
Load (id)
Load (id+b)
Load (id+c)
Load (id+d)
```

```
__kernel void vector(
    float4 *input,
    int b, int c, int d)
{
    unsigned int id = get_global_id(0)

    float4 a = (float4)(
        input[id],
        input[id+b],
        input[id+c],
        input[id+d],
    )

    …
}
```

11. You have written a program that sometimes produces the wrong result. After adding some lock primitives to your program, you cannot notice any problems. What is the likely reason for why your original program produced wrong results? Provide an example, of your own, explaining your problem.

As in the program when there would be sharable resources or there would be critical section in the code which is shared among two or more processes then it will create race problem which will produce wrong results. So it is required that critical section of the code should be accessed by only one process at a time otherwise it will create race condition and produce undesirable results as we can see in case of bank account example where banks have several cash machines and without use of lock , if we see this problem then , there would be bank account in which withdrawal and deposit are two function which helps to withdraw and deposit one dollar. Here we are providing depositing the 1 dollar cash with withdrawal then after whole day it will leave 0 bank balance as depositing would be followed by withdrawal of 1 dollar. But if we run code then it will not give output as 0 as due to interleaving it will produce wrong results. If two cash machines X and Y will deposit money at same time then it will produce wrong result due to interleaving. So this would become the example of race condition. So with the help of lock primitives we can achieve desirable results as with the help of lock primitive in critical section , only one process have access at a time. So after completing whole code written in lock primitive and comes out from critical section then other process will enter into critical section which will help to avoid race condition.

12. In what situations does it make sense to use barriers in parallel programs? Provide one example of your own

Barrier in parallel programming is type of synchronisation method in which barrier for a group of thread means that at a certain point all thread stops executing and wait till all the threads will not reach at this point. Here synchronisation barriers cannot be shared across the processes. Barriers in multiprogramming helps to achieve synchronisation easily without providing multiple conditions and lock and mutex. So it would be an easy method to achieve synchronisation among various threads.To initialize synchronisation barrier we can use pthread_barrier_init () to allocate resources required to use the barrier. Example of barrier in parallel programming would be parallel do loop in FORTRAN with OpenMP which will not allow any thread to move further until all the threads completed its iterations.

13. Fork-join is useful in many parallel programs. Briefly mention two problems where fork-join is useful.

Fork join is used in parallel computing in which a problem is divided into subproblems and then execute it and merge at a subsequent point and resume sequential execution.
Two problems where fork join would be useful as-
1. Fork join model would be useful in merge sort problem of CLRS which are -

```
mergesort(arr,start,end):
if(start<end):
midvalue=(start+(end-start)/2)
fork mergesort(arr,start,midvalue)
mergesort(arr,midvalue,high)
Join
merge(arr,start,midvalue,high)
```

So as we can see here mergesort works upon fork join model in which first recursive call will run in parallel in separate thread with that part of function then join helps to synchronize all threads. It helps to improve the capacity of machines to a great extent.

Fork join framework would be used in JAVA 7 as it provide tools which helps to achieve parallel processing with use of all available processor cores which would be completed through divide and conquer approach as it will break down a process into several task and then allow them to execute and at a point it will merge all the results of individual parts which will reduce computation time and improve the efficiency of program.

14. The client-server pattern is fundamental to many distributed systems. Explain with an example, how a client makes use of the client-server pattern.

    Distributed system is the system in which multiple computers are connected to each other to solve a single problem . As that problem would be divided into various subproblems and each subproblem would be given to each computer which will improve the efficiency of program and reduce computational time. Client server pattern will have two main components which are client and server in which request would be sent by the client to server and server responds back the client. Client server network can connect multiple users and provide various security measures. There would be various advantages of client server model in distributed system as client and server capacity would be changed separately and it is more cost efficient.

    Example where client makes use of client - server pattern in distributed system is web browsers as web browsers act as a client which will request to various servers for displaying the content of various web pages. Here web browsers will send HTTP request to the server and connection would be done using TCP/IP.

15. What are the advantages and disadvantages of IaaS, Paas, and SaaS respectively?

16. Explain in your own words what the differences are between temporal and spatial locality.

17. Reductions are commonly used with barriers, either explicit or implicit barriers. Why are reductions used with barriers? Give an example of a situation when barriers and reductions must be used together.

    Barriers and reductions are global operation used in large programs. To improve the performance of the programs Barriers and reductions are used.

A barrier is a synchronization method. A barrier is a point in the program logic which stops the program to execute further untill all the threads in that program reach that point.
Reduction is the process of reducing the complexity of the problem. It will generate subprograms which are easier to solve or for which solution is already existing. It reuses the already existing program which is the sub-program of the complex incoming program(this will be reduced to sub-programs) which helps in time-consumption and compilation.when reductions are used in program, sub-programs get generated and will run in parallel which helps the program to reach the barrier within less-time increasing the performance.
example: Improve OpenMP Performance using barriers and reductions,AI,Reducing the Barrier to Entry of Complex Robotic Software
refer wikipedia for brief explanation of the above examples
Ingeneral one prefer to use barriers and reductions when the code is getting complex,to increase the performance of code,to reduce the compilation time.

Question 1

Today, what are the fundamental limits to the execution speed of a single CPU?

We've begun to reach the physical limitation for the:

- Size (transistor size on CPU)
- Cooling (a certain amount of power is converted to heat, and despite new technologies which greatly optimise it, we're still reaching the limit)

Question 2

Explain how caches can reduce the average memory access time.

Accessing the memory takes a very long time, but the bus length to transfer data can be rather large in comparison, so by taking a LOT of data and dumping in the cache, allows for faster access on average, significantly.

There are different policies about what to include, and when to change it,

Common examples for 'Cache Replacement policies':

First in First out (FIFO)

Last in First out (LIFO)

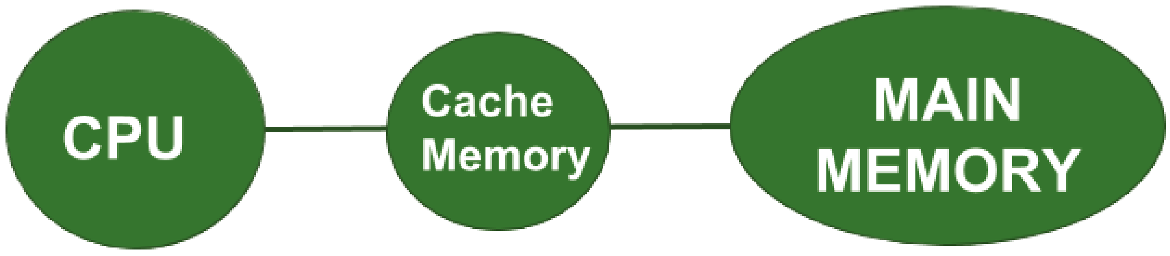
Least-frequently Used (LFU)

Most recently used (MRU)

Source list: https://en.wikipedia.org/wiki/Cache_replacement_policies

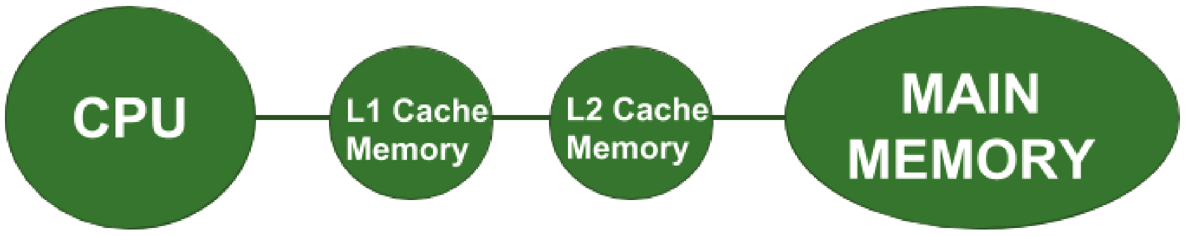
Cache is a random access memory used by the CPU to reduce the average time taken to access memory.
Multilevel Caches is one of the techniques to improve Cache Performance by reducing the *"MISS PENALTY"*. Miss Penalty refers to the extra time required to bring the data into cache from the Main memory whenever there is a ***"miss"*** in the cache.
For clear understanding let us consider an example where the CPU requires 10 Memory References for accessing the desired information and consider this scenario in the following 3 cases of System design :

Case 2 : System Design with Cache Memory



Here the CPU at first checks whether the desired data is present in the Cache Memory or not i.e. whether there is a ***"hit"*** in cache or ***"miss"*** in the cache. Suppose there is *3* miss in Cache Memory then the Main Memory will be accessed only 3 times. We can see that here the miss penalty is reduced because the Main Memory is accessed a lesser number of times than that in the previous case.

Case 3 : System Design with Multilevel Cache Memory



Here the Cache performance is optimized further by introducing multilevel Caches. As shown in the above figure, we are considering 2 level Cache Design. Suppose there is *3 miss* in the L1 Cache Memory and out of these 3 misses there is *2 miss* in the L2 Cache Memory then the Main Memory will be accessed only 2 times. It is clear that here the Miss Penalty is reduced considerably than that in the previous case thereby improving the Performance of Cache Memory.
NOTE :
We can observe from the above 3 cases that we are trying to decrease the number of Main Memory References and thus decreasing the Miss Penalty in order to improve the overall System Performance. Also, it is important to note that in the Multilevel Cache Design, L1 Cache is attached to the CPU and it is small in size but fast. Although, L2 Cache is attached to the Primary Cache i.e. L1 Cache and it is larger in size and slower but still faster than the Main Memory.

Instead of a brute-force approach of making all memory fast processor designer include a cache mechanism in the processor which is checked to see if a copy of a particular memory location is available in the cache before initiating the actual memory access. If the cache holds a copy of the memory location, the slow access to main memory can be skipped. The cache is small and fast, usually tens or hundreds of kilobytes in size accessible in just a few clock cycles.

Question 3

How can instruction level parallelism improve execution speed? What are the main approaches for implementing instruction level parallelism?

Instruction-level parallelism, or ILP, attempts to **improve processor performance by having multiple processor components or functional units simultaneously executing instructions. There are two main approaches to ILP: pipelining, in which functional units are arranged in stages, and multiple issue, in which multiple instructions can be simultaneously initiated. Both approaches are used in virtually all modern CPUs.**
Pipelining The principle of pipelining is similar to a factory assembly line: while one team is bolting a car's engine to the chassis, another team can connect the transmission to the engine and the driveshaft of a car that's already been processed by the first team, and a third team can bolt the body to the chassis in a car that's been processed by the first two teams. As an example involving computation, suppose we want to add the floating point numbers 9.87×104 and 6.54×103. Then we can use the following steps:

You do more in less time. The concept is that you're running multiple processes/nodes at the same time, working tasks that can be handled in parallel, thus improving the overall execution speed.

Source: page 25 in the pacheco book 2.2.5 Instruction-level parallelism Instruction-level parallelism, or ILP, attempts to improve processor performance by having multiple processor components or functional units simultaneously executing instructions. There are two main approaches to ILP: pipelining, in which functional units are arranged in stages, and multiple issue, in which multiple instructions can be simultaneously initiated. Both approaches are used in virtually all modern CPUs.

Instruction-level parallelism (ILP): This type of parallelism is trying to exploit the potential overlap between instructions in a computer program. Most forms of ILP are implemented and realized on each processor's hardware:

Instruction Pipelining —Execute different stages of different independent Instructions in the same cycle thus making full use of idle resources.

Superscalar Processors — Utilize multiple execution units in a single processor die, thus following instructions can be executed without waiting on complex preceding instructions to finish executing.

Out-of-order execution — Instructions not violating any data dependencies are executed when a unit is available even when preceding instructions are still executed.

Speculative execution / Branch prediction — The processor tries to avoid stalling while control instructions (branches — i.e. if or case commands) are still resolved by executing the most probable flow of the program.

Most processors use a combination of the above ILP techniques. For example, Intel Pentium series used all of the above techniques to achieve higher performance with each product generation. Static ILP parallelism on the other hand, can be achieved on software level by specialized compilers which are used by the specialized *Very long instruction word* (VLIW) processors. VLIW processors have multiple execution units organized in multiple pipelines and require the compilers to prepare the parallel instruction streams to fully utilize their resources.

Task / Thread-level parallelism: This high-level form of parallel computing is focusing on partitioning the application to be executed in distinct *tasks* or *threads,* that can be then executed simultaneously on different computation units. Threads can work on independent data fragments or even share data between them.

The developer has to use the *multi-threaded programming* paradigm in order to fully utilize the capabilities of the nowadays available multicore processors. This new programming paradigm requires a shift of mentality while programming new applications, which is somewhat difficult, as until recently programming was done *sequentially* (a single-thread holding the whole application).

Modern Operating Systems have been used to provide a transparent utilization of all of the available cores of modern multicore processors by scheduling different processes on different cores. However, this is not the case for executing efficiently complex bioinformatics applications, because the computation load of each process cannot be distributed on the available cores by the OS. As such, these applications must be re-developed with multi-threading in mind in order to achieve higher level of parallelism though thread-level parallelism and improve their performance.

Data parallelism:A form of high-level parallelism that in contrast to thread-level parallelism tries to partition the data to different available computation units instead. The cores execute the *same* task code over the data assigned to each. This form requires advanced developing skills to achieve, as the code executed should be executed independently on each data fragment. In addition, it is applicable to specific problems only.

Data parallelism is the only available option for high-level parallelism in computer graphics because the graphic processor units (GPUs) are designed to execute each graphic processing task as fast as possible by partitioning each frame in regions. The task on command is then executed independently on each data region by their hundreds of processing units.

Question 4

Give one example of each type of machine in Flynn's taxonomy.

SISD (LC-3)

SIMD (Parallel computing such as multithreading)

MISD (Multiple computers having to agree with the same data, example space shuttle flights)

MIMD (Distributed systems (i.e. networking))

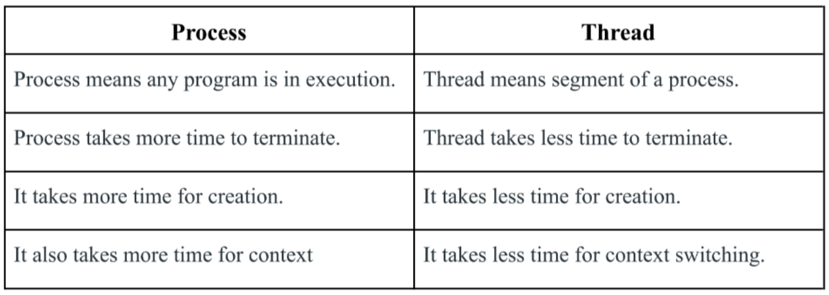
Question 5

[Repetition from 62588] What is a process? What is a thread? What are the main differences and similarities between the two?

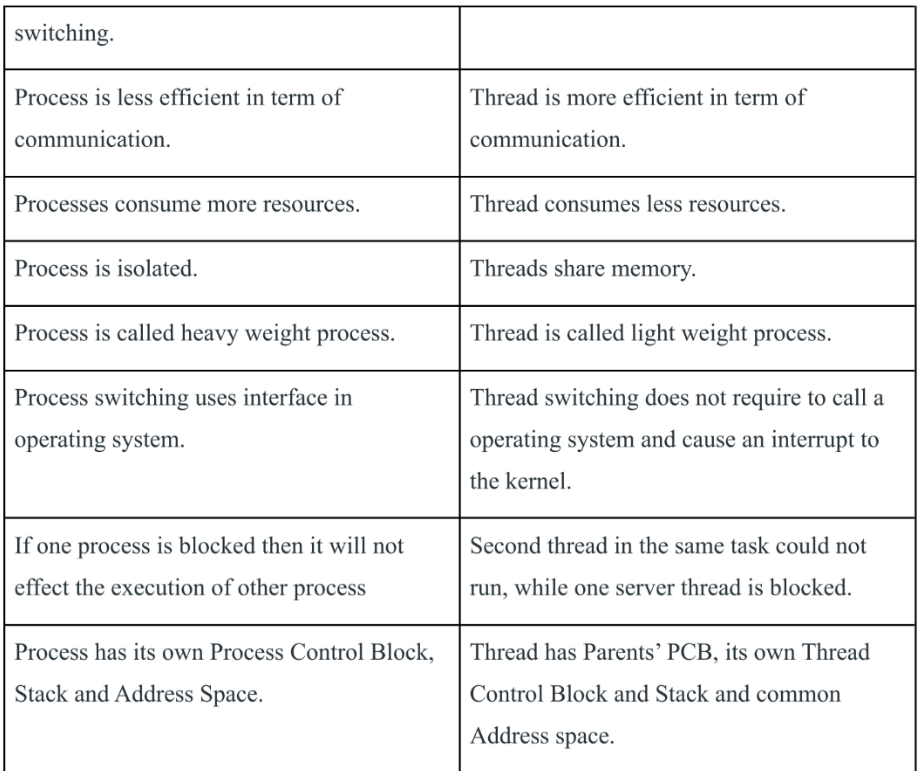
Thread is an execution unit that consists of its own program counter, a stack, and a set of registers where the program counter mainly keeps track of which instruction to execute next, a set of registers mainly hold its current working variables, and a stack mainly contains the history of execution
Threads are also known as Lightweight processes. Threads are a popular way to improve the performance of an application through parallelism. Threads are mainly used to represent a software approach in order to improve the performance of an operating system just by reducing the overhead thread that is mainly equivalent to a classical process.
The CPU switches rapidly back and forth among the threads giving the illusion that the threads are running in parallel. As each thread has its own independent resource

for process execution; thus Multiple processes can be executed parallelly by increasing the number of threads.
It is important to note here that each thread belongs to exactly one process and outside a
process no threads exist. Each thread basically represents the flow of control separately.
process is a software program that is currently running in Windows. Every process has an ID, a number that identifies it. A thread is an object that identifies which part of the program is running. Each thread has an ID, a number that identifies it.
A process may have more than one thread. The purpose of a thread is to allocate processor time. On a machine with one processor, more than one thread can be allocated, but only one thread can run at a time. Each thread only runs a short time and then the execution is passed on to the next thread, giving the user the illusion that more than one thing is happening atonce. On a machine with more than one processor, true multithreading can take place.

Process vs Thread



Process	Thread
Process means any program is in execution.	Thread means segment of a process.
Process takes more time to terminate.	Thread takes less time to terminate.
It takes more time for creation.	It takes less time for creation.
It also takes more time for context	It takes less time for context switching.



switching.	
Process is less efficient in term of communication.	Thread is more efficient in term of communication.
Processes consume more resources.	Thread consumes less resources.
Process is isolated.	Threads share memory.
Process is called heavy weight process.	Thread is called light weight process.
Process switching uses interface in operating system.	Thread switching does not require to call a operating system and cause an interrupt to the kernel.
If one process is blocked then it will not effect the execution of other process	Second thread in the same task could not run, while one server thread is blocked.
Process has its own Process Control Block, Stack and Address Space.	Thread has Parents' PCB, its own Thread Control Block and Stack and common Address space.

Process: Executable program run on a processor

Thread: Can run as part of a program, as a thread on a core

Question 6

[Repetition from 62588] What is a race condition? How can you remove a race condition from your program?

When multiple processes/threads access the same data at the same time, which can cause issues.

Such as the variable 'derp' (= 10) is accessed from 20 different sources, at the same time, each of them adding 1, and then saving the result. At the end, the final value in 'derp' is 11.

Solved by utilizing: Mutex & Semaphores

Question 7

What is the speedup of a parallel program?

Speedup achieved by a parallel algorithm is defined as **the ratio of the time required by the best sequential algorithm to solve a problem, T(1), to the time required by parallel algorithm using p processors to solve the same problem,T(p)**.

Serial / parallel

S = (T_serial / T_parallel)

So time in serial over parallel.

Question 8

What is the main message of Amdahl's law?

Amdahl's law: *Amdahl's law is actually modelling the expected improvement of parallelized implementations of an already existing algorithm implementation. The execution time of the parallel version is limited by the percentage of the sequential (non-parallelized) code. For example, if 90% of an algorithm is parallelizable, the overall improvement can be no more than 10x. An assumption made by Amdahl's law is that the problem size remains the same when the problem is parallelized. However, this is not the norm and Amdhal's law just describes the best-case scenario. Splitting data for processing between many different processing units, results in I/O and memory overheads. These overheads negatively affect overall performance.*

Use it to find the theoretical max speedup, of a combined serial/parallel program.

Question 9

What are the main steps in writing a parallel program, according to Foster?

1. Partition the problem solution into tasks.

2. Identify the communication channels between the tasks.

3. Aggregate the tasks into composite tasks. (combine stuff)

4. Map the composite tasks to core

Question 10

Measuring execution time can be challenging. List some problems you may face when measuring time in a program.

Scheduling policies (OS)

Communication between nodes

Faulty clock cycle

Speed can be different on different computers (i.e. hardware specs, processor and RAM)

Question 11

Why do we need to start an MPI program with MPI_Init and end it with MPI_Finalize?

The init and finalize is to create and close a communication method between the nodes.

Question 12

In MPI, what is a communicator?

Holds a group of processes that can communicate with each other. All message passing calls in MPI must have a specific communicator to use the call with. An example of a communicator handle is MPI_COMM_WORLD. MPI_COMM_WORLD is the default communicator that contains all processes available for use

Question 13

What is the point of collective communication in MPI? Can't all

possible communication patterns be implemented with point-to-point

communication primitives?

Technically yes, it's possible. But it's highly inefficient, so tl;dr Superflous steps, & 'Efficiency

Question 14

In MPI, there is an MPI_Barrier primitive? What does it do? Isn't it

odd for MPI to have such a primitive? In what situations is it useful?

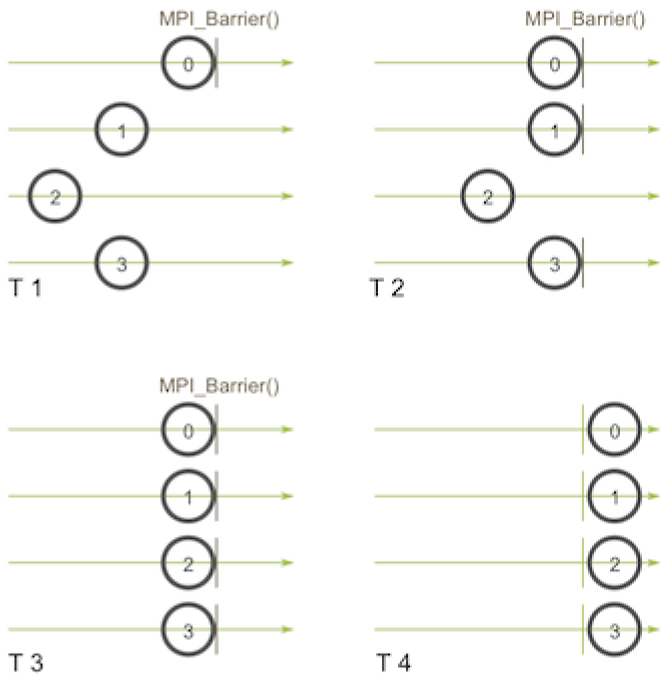
Barrier is a function, that ensures that all nodes reach the same point and stop. It can be useful to prevent one or more nodes progressing beyond a point where the other nodes haven't processed the data to be shared.

One of the things to remember about collective communication is that it implies a *synchronization point* among processes. This means that all processes must reach a point in their code before they can all begin executing again.

Before going into detail about collective communication routines, let’s examine synchronization in more detail. As it turns out, MPI has a special function that is dedicated to synchronizing processes:

```
MPI_Barrier(MPI_Comm communicator)
```

The name of the function is quite descriptive - the function forms a barrier, and no processes in the communicator can pass the barrier until all of them call the function. Here’s an illustration. Imagine the horizontal axis represents execution of the program and the circles represent different processes:



Process zero first calls `MPI_Barrier` at the first time snapshot (T 1). While process zero is hung up at the barrier, process one and three eventually make it (T 2). When process two finally makes it to the barrier (T 3), all of the processes then begin execution again (T 4).

`MPI_Barrier` can be useful for many things. One of the primary uses of `MPI_Barrier` is to synchronize a program so that portions of the parallel code can be timed accurately.

Question 15

What does it mean to parallelize code with OpenMP?

OpenMP is a 'shared memory' parallel programming model. Meaning that you can use 'OpenMP' to turn parts of a serial code into a parallel version. 'OpenMP' is a specific library which can implement this function with a higher abstraction level.

Question 16

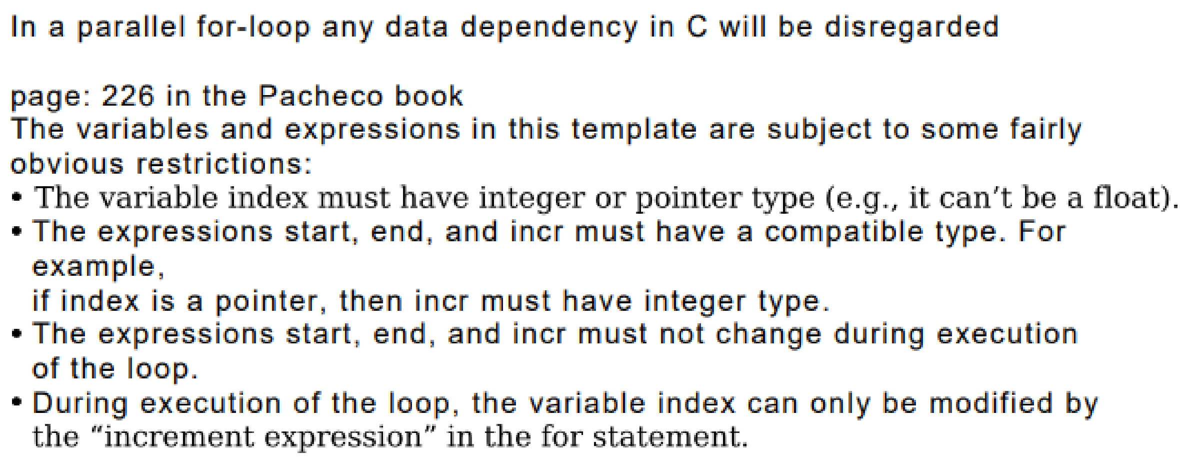
Why are #pragma directives needed in OpenMP?

They're used to communicate between the different process threads Such as running specific code chunks in parallel, or even atomic operations. Concept is about having a higher abstraction level, for multiple threads at the same time

Question 17

Why can you not break out of a parallel for loop in OpenMP using return or a goto?

tl;dr for-loops in OpenMP are rather restrictive, because they're working on same data, so even if one of the threads break/return, the others won't (even if they're supposed to).


In a parallel for-loop any data dependency in C will be disregarded

page: 226 in the Pacheco book
The variables and expressions in this template are subject to some fairly obvious restrictions:

- The variable index must have integer or pointer type (e.g., it can't be a float).
- The expressions start, end, and incr must have a compatible type. For example,
  if index is a pointer, then incr must have integer type.
- The expressions start, end, and incr must not change during execution of the loop.
- During execution of the loop, the variable index can only be modified by the "increment expression" in the for statement.

Question 18

Why are there different types of OpenMP scheduling?

Different scheduling types have different runtimes, so it's important to choose the type that would fit your scenario the best.

Static

Dynamic

auto/guided

runtime

If you don't define it within the program it merely use a 'default' scheme (uncertain of specific formatting)

Question 19

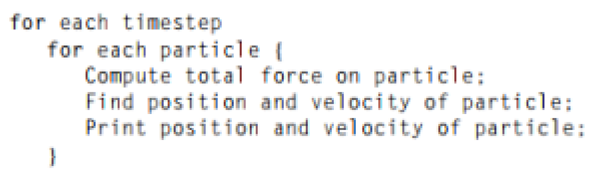
How do you define a critical section in OpenMP?

For sections that require preventions of race conditions, use 'critical' directive.
# pragma omp critical
This forces the threads to run in serial, rather than parallel, for that specific section.

Question 20

Solve problem 6.1 in Pacheco.

6.1 In each iteration of the serial n-body solver, we first compute the total force on each particle, and then we compute the position and velocity of each particle. Would it be possible to reorganize the calculations so that in each iteration we did all of the calculations for each particle before proceeding to the next particle? That is, could we use the following pseudocode?



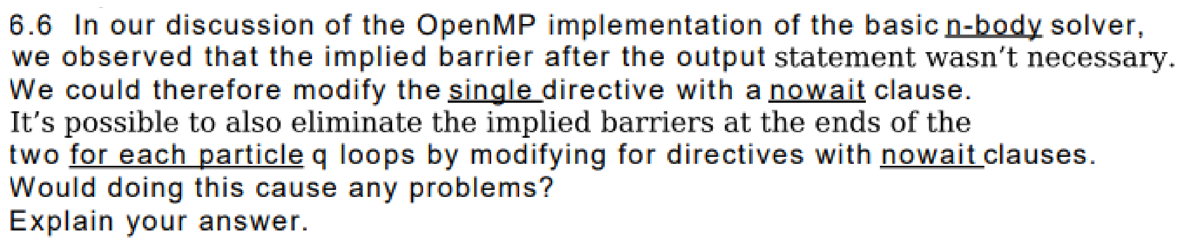
```
for each timestep
   for each particle {
      Compute total force on particle;
      Find position and velocity of particle;
      Print position and velocity of particle;
   }
```

If so, what other modifications would we need to make to the solver? If not, why not?

If there is data dependency then no, else yes. I can't provide a proper answer based on that piece of pseudo code

Question 21

Solve problem 6.6 in Pacheco.



6.6 In our discussion of the OpenMP implementation of the basic n-body solver, we observed that the implied barrier after the output statement wasn't necessary. We could therefore modify the single directive with a nowait clause. It's possible to also eliminate the implied barriers at the ends of the two for each particle q loops by modifying for directives with nowait clauses. Would doing this cause any problems? Explain your answer.

Question 22

Compare a parallel and distributed system. What are the differences in purpose between the two types of systems?

Parallel system is mainly created in such a way that multiple processes/threads can actively worn on the same problem in parallel, thus increasing efficiency. For distributed systems, each 'node' can work independently of one another, on multiple different problems. Obviously it can also work as a 'parallel' program, but it would require some additional communication between the nodes to achieve this

Both parallel and distributed computing have been around for a long time and both have contributed greatly to the improvement of computing processes. However, they have key differences in their primary function.

Parallel computing, also known as parallel processing, speeds up a computational task by dividing it into smaller jobs across multiple processors inside one computer. Distributed computing, on the other hand, uses a distributed system, such as the internet, to increase the available computing power and enable larger, more complex tasks to be executed across multiple machines.

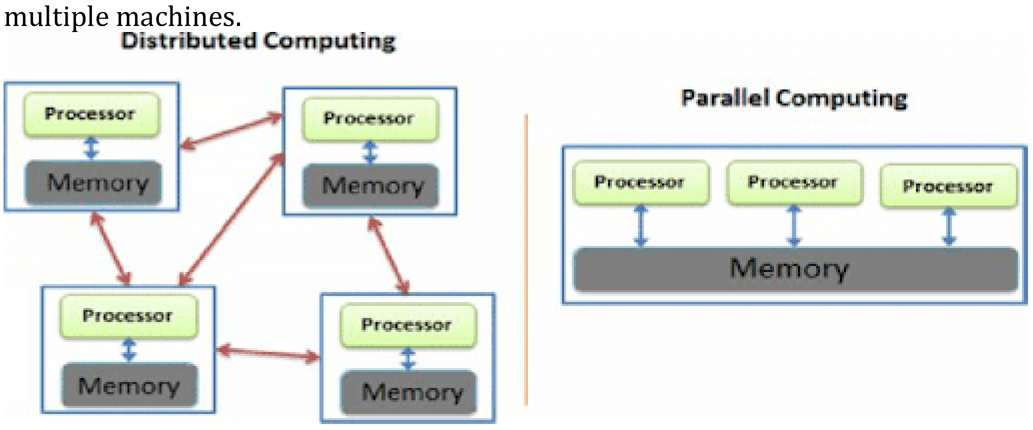


Figure 1: A distributed computing system compared to a parallel computing system. *Source: ResearchGate*

What Is Parallel Computing?

Parallel computing is the process of performing computational tasks across multiple processors at once to improve computing speed and efficiency. It divides tasks into sub-tasks and executes them simultaneously through different processors.

There are three main types, or "levels," of parallel computing: bit, instruction, and task.

- **Bit-level parallelism:** Uses larger “words,” which is a fixed-sized piece of data handled as a unit by the instruction set or the hardware of the processor, to reduce the number of instructions the processor needs to perform an operation.
- **Instruction-level parallelism:** Employs a stream of instructions to allow processors to execute more than one instruction per clock cycle (the oscillation between high and low states within a digital circuit).
- **Task-level parallelism:** Runs computer code across multiple processors to run multiple tasks at the same time on the same data.

What Is Distributed Computing?

Distributed computing is the process of connecting multiple computers via a local network or wide area network so that they can act together as a single ultra-powerful computer capable of performing computations that no single computer within the network would be able to perform on its own.

Distributed computers offer two key advantages:

- **Easy scalability:** Just add more computers to expand the system.
- **Redundancy:** Since many different machines are providing the same service, that service can keep running even if one (or more) of the computers goes down.

Parallel Computing: A Brief History

Etchings of the first parallel computers appeared in the 1950s when leading researchers and computer scientists, including a few from IBM, published papers about the possibilities of (and need for) parallel processing to improve computing speed and efficiency. The 1960s and ’70s brought the first supercomputers, which were also the first computers to use multiple processors.

The motivation behind developing the earliest parallel computers was to reduce the time it took for signals to travel across computer networks, which are the central component of distributed computers.

Parallel computing went to another level in the mid-1980s when researchers at Caltech started using massively parallel processors (MPPs) to create a supercomputer for scientific applications.

Today, multi-core processors have made it mainstream, and the focus on energy efficiency has made using parallel processing even more important, as increasing performance through parallel processing tends to be much more energy-efficient than increasing microprocessor clock frequencies.

Key Differences Between Parallel Computing and Distributed Computing

While parallel and distributed computers are both important technologies, there are several key differences between them.

Difference #1: Number of Computers Required

Parallel computing typically requires one computer with multiple processors. Distributed computing, on the other hand, involves several autonomous (and often geographically separate and/or distant) computer systems working on divided tasks.

Difference #2: Scalability

Parallel computing systems are less scalable than distributed computing systems because the memory of a single computer can only handle so many processors at once. A distributed computing system can always scale with additional computers.

Difference #3: Memory

In parallel computing, all processors share the same memory and the processors communicate with each other with the help of this shared memory. Distributed computing systems, on the other hand, have their own memory and processors.

Difference #4: Synchronization

In parallel computing, all processors share a single master clock for synchronization, while distributed computing systems use synchronization algorithms.

Difference #5: Usage

Parallel computing is used to increase computer performance and for scientific computing, while distributed computing is used to share resources and improve scalability.

When to Use Parallel Computing: Examples

This computing method is ideal for anything involving complex simulations or modeling. Common applications for it include seismic surveying, computational astrophysics, climate modeling, financial risk management, agricultural estimates, video color correction, medical imaging, drug discovery, and computational fluid dynamics.

When to Use Distributed Computing: Examples

Distributed computing is best for building and deploying powerful applications running across many different users and geographies. Anyone performing a Google search is already using distributed computing. Distributed system architectures have shaped much of what we would call “modern business,” including cloud-based computing, edge computing, and software as a service (SaaS).

Which Is Better: Parallel or Distributed Computing?

It’s hard to say which is “better”—parallel or distributed computing—because it depends on the use case (see section above). If you need pure computational power and work in a scientific or other type of highly analytics-based field, then you’re probably better off with parallel computing. If you need scalability and resilience and can afford to support and maintain a computer network, then you’re probably better off with distributed computing.

Question 23

What is the role of the middleware?

To assist the development of distributed applications, distributed systems are often organised to have a separate layer of software that is logically placed on top of the respective operating systems of the computers that are part of the system. In a sense, middleware is the same to a distributed system as what an operating system is to a computer: a manager of resources offering its applications to efficiently share and deploy those resources across a network. Next to resource management, it offers services that can also be found in most operating systems, including:

- Facilities for interapplication communication
- Security services
- Accounting services
- Masking if and recovery from failures

The main difference with their operating-system equivalents, is that middleware services are offered in a networked environment. Note also that most services are useful to many applications. In this sense, middleware can also be viewed as a container of commonly used components and functions that now no longer have to be implemented by applications separately.

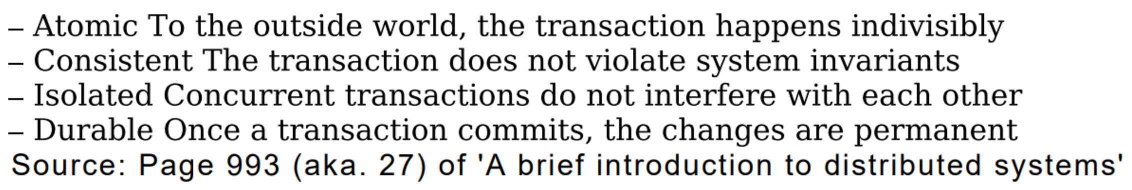
Question 24

What are the pros and cons of IaaS, Paas, and SaaS respectively?

Question 25

Explain the ACID properties and how they can be maintained.

This all-or-nothing property of transactions is one of the four characteristic properties that transactions have. More specifically, transactions adhere to the so-called ACID properties:



– Atomic To the outside world, the transaction happens indivisibly
– Consistent The transaction does not violate system invariants
– Isolated Concurrent transactions do not interfere with each other
– Durable Once a transaction commits, the changes are permanent
Source: Page 993 (aka. 27) of 'A brief introduction to distributed systems'

This allows a way to validate a transaction from the time of data input to the final output, which is stored in a database

Make use of middleware to 'maintain' and support the functions.

In distributed systems, transactions are often constructed as a number of sub transactions, jointly forming a nested transaction as shown in Fig. 9. The top-level transaction may fork off children that run in parallel with one another, on different machines, to gain performance or simplify programming.

This allows a way to validate a transaction from the time of data input to the final output, which is stored in a database Make use of middleware to 'maintain' and support the functions.

Question 26

What are the high-level building blocks for a warehouse-scale computer?

Specific hardware (chipsets, storage)

Networking infrastructure (communication between racks, aka network switches)

Cooling capacity and associated hardware

Power usage

Question 27

What is churn?

"Workload churn" Most common term is the 'rate of members of a group leaving the group' Which in this case refers to how many members are leaving the service, thus negatively impacting the amount of work required

Question 28

Communication is costly in many systems, including IoT system. How

can the computing continuum reduce communication and its cost?

By running lots of processes along the edge

** Computing continum makes the process of communication as automated. By this the communication becomes simpler and it helps to reduce the cost of communication, since the automation helps in providing replies for normal queries without any involvement of live feed.

** Example of computing continum is Chat Bot which helps to provide solutions to our queries with the process of automation. This is how computing continum helps to reduce the communication and it's cost.

** The systems that are part of this computing continum are distributed and non-closed systems, since they helps in automation.

I hope the answer is clear. Thank you.

Question 29

What type of systems can be part of a computing continuum?

Edge applications that efficiently process data quickly are proliferating as a result of dramatic shifts in the technology landscape characterized by growing sizes and the pervasiveness of computing and data. New classes of applications that seamlessly integrate real-time data with complex models and data analytics to monitor and manage systems of interest are being investigated as the levels and fidelity of instrumentation rise and the types and volumes of available data expand. To support dynamic and data-driven application workflows, these applications must, nevertheless, seamlessly integrate resources at the edge, the core, and throughout the data stream; in other words, they must use a computing continuum.

Please give upvote and if you have any doubt ask in comment.

Exercises

Problem 1

Why are failure rates in warehouse-scale computers high even though each sub-component is fairly reliable?

In warehouse-scale computers (WSCs), however, hardware cannot easily be made “reliable enough” as a result of the scale. Suppose a cluster has ultra-reliable server nodes with a stellar mean time between failures (MTBF) of 30 years (10,000 days)—well beyond what is typically possible to achieve at a realistic cost. Even with these ideally reliable servers, a cluster of 10,000 servers will see an average of one server failure per day. Large and complex Internet services are often composed of several software modules or layers that are not bug-free and can themselves fail at even higher rates than hardware components. Providing an accurate quantitative assessment of faults in WSC systems would be challenging given the diversity of equipment and software infrastructure across di"erent deployments.

Service Level Failures

We broadly classify service-level failures into the following categories, listed in decreasing degree of severity: R5

Corrupted: committed data that are impossible to regenerate, are lost, or corrupted.

Unreachable: service is down or otherwise unreachable by the users, R5
Degraded: service is available but in some degraded mode.

Masked: faults occur but are completely hidden from users by the fault-tolerant software/hardware mechanisms

Problem 2

In warehouse-scale computers, a significant effort is spent on automatic service monitor-ing. Why, would you say, is that so?

Because the size and complexity of both the workloads and the hardware infrastructure make the monitoring framework a fundamental component of any such deployments.

The monitoring information must be very fresh to let an operator (or an automated system) take corrective actions quickly and avoid signi%cant disruption—within seconds, not minutes. Fortunately, the most critical information needed is restricted to just a few signals that can be collected from the front-end servers, such as latency and throughput statistics for user requests. In its simplest form, such a monitoring system could be simply a script that polls all front-end servers every few seconds for the appropriate signals and displays them to operators in a dashboard.

Large-scale services often need more sophisticated and scalable monitoring support, as the number of front-ends can be quite large, and more signals are needed to characterize the health of the service. For example, it may be important to collect not only the signals themselves but also their derivatives over time. !e system may also need to monitor other business-speci%c parameters in addition to latency and throughput. !e monitoring system may need to support a simple language that lets operators create derived parameters based on baseline signals being monitored. Finally, the system may need to generate automatic alerts to on-call operators depending on monitored values and thresholds. Getting a system of alerts (or alarms) well tuned can be tricky because alarms that trigger too often because of false positives will cause operators to ignore real ones, whereas alarms that trigger only in extreme cases might get the operator’s attention too late to allow smooth resolution of the underlying issues.

Problem 3

Tail latency is a significant problem in warehouse-scale computers? Why is that so?

Reducing the long tail of the query latency distribution in modern warehouse scale computers is critical for improving performance and quality of service (QoS) of workloads such as Web Search and Memcached. Traditional turbo boost increases a processor’s voltage and frequency during a coarse-grained sliding window, boosting all queries that are processed during that window. However, the inability of such a technique to pinpoint tail queries for boosting limits its tail reduction benefit. In this work, we propose Adrenaline, an approach to leverage finer-granularity (tens of nanoseconds) voltage boosting to effectively rein in the tail latency with query-level precision.

If an individual leaf node has only a 1-in-100 chance of exceeding a 1s response time, when aggregating parallel requests to 100 nodes, 63% of queries will take longer than 1s. Due to this service-level sensitivity to leaf-node tail latency, optimizations to reduce the long tail are paramount. Indeed, sacrificing mean latency for improved tail latency is encouraged.

What is IaaS, PaaS and SaaS?

The cloud is a broad concept embracing different sorts of online services. For those who consider cloud services for their business, it’s important to grasp the difference between IaaS, PaaS and SaaS — the core cloud models available. You should choose the particular model depending on your business requirements and on the number of tasks you want to perform yourself or delegate to the service provider.

The -aaS acronyms are always confusing. They mean “something” as a service. Today, practically everything can be presented as a service. One of the most popular questions is, what is the difference between IaaS, PaaS and SaaS? In our article, we want to have the terms SaaS, PaaS, IaaS explained in order to help you develop the right understanding of the concept and create a suitable cloud migration strategy for your organization.

- **IaaS** — Infrastructure as a Service
- **PaaS** — Platform as a Service
- **SaaS** — Software as a Service

These are three basic models of the cloud service. Each one has its own benefits and peculiarities.

Read also: Everything-as-a-Service (XaaS): Definitions and Examples

IaaS: Infrastructure as a Service

This is a virtual equivalent of a traditional data center. Cloud infrastructure providers use virtualization technology to deliver scalable compute resources such as servers, networks and storage to their clients. This is beneficial for the clients, as they don’t have to buy personal hardware and manage its components. Instead, they can deploy their platforms and applications within the provider’s virtual machines that offer the same technologies and capabilities as a physical data center.

An IaaS provider is responsible for the entire infrastructure, but users have total control over it. In turn, users are responsible for installing and maintaining apps and operating systems, as well as for security, runtime, middleware and data.

IaaS Key Features

- Highly scalable resources
- Enterprise-grade infrastructure
- Cost depends on consumption
- Multitenant architecture, i.e. a single piece of hardware serves many users
- The client gets complete control over the infrastructure

IaaS Advantages

- The most flexible and dynamic model

- Cost-effective due to pay-as-you-go pricing
- Easy to use due to the automate d deployment of hardware
- Management tasks are virtualized, so employees have more free time for other tasks

IaaS Disadvantages

- Data security issues due to multitenant architecture
- Vendor outages make customers unable to access their data for a while
- The need for team training to learn how to manage new infrastructure

When to Use IaaS

IaaS can be especially advantageous in some situations:

- If you are a small company or a startup that has no budget for creating your own infrastructure
- If you are a rapidly growing company and your demands are unstable and changeable
- If you are a large company that wants to have effective control over infrastructure but pay only for the resources you actually use

Examples of IaaS

The best-known IaaS solution s vendors are Microsoft Azure, Google Compute Engine (GCE), Amazon Web Services (AWS), Cisco Metapod, DigitalOcean, Linode and Rackspace.

Read also: 4 Best Cloud Deployment Models You Need to Know

PaaS: Platform as a Service

PaaS in cloud computing is a framework for software creation delivered over the internet. This is the offering of a platform with built-in software components and tools, using which developers can create, customize, test and launch applications. PaaS vendors manage servers, operating system updates, security patches and backups. Clients focus on app development and data without worrying about infrastructure, middleware and OS maintenance.

The main difference between IaaS and PaaS lies in the degree of control given to users.

PaaS Key Features

- Allows for developing, testing and hosting apps in the same environment
- Resources can be scaled up and down depending on business needs
- Multiple users can access the same app in development
- The user doesn’t have complete control over the infrastructure
- Web services and databases are integrated
- Remote teams can collaborate easily

PaaS Advantages

- PaaS-built software is highly scalable, available and multi-tenant, as it is cloud-based
- The development process is quickened and simplified
- Reduced expenses for creating, testing and launching apps
- Automated company policy
- Reduced amount of coding required
- Allows for easy migrating to the hybrid cloud

PaaS Disadvantages

- Data security issues
- Compatibility of existing infrastructure (not every element can be cloud-enabled)
- Dependency on vendor’s speed, reliability and support

When to Use PaaS

Such solutions are especially profitable to developers who want to spend more time coding, testing and deploying their applications. Utilizing PaaS is beneficial when:

- Multiple developers work on one project
- Other vendors must be included
- You want to create your own customized apps

Examples of PaaS

The best-known PaaS solutions vendors are Google App Engine, Amazon AWS, Windows Azure Cloud Services, Heroku, AWS Elastic Beanstalk, Apache Stratos and OpenShift.

SaaS: Software as a Service

With this offering, users get access to the vendor’s cloud-based software. Users don’t have to download and install SaaS applications on local devices, but sometimes they may need plugins. SaaS software resides on a remote cloud network and can be accessed through the web or APIs. Using such apps, customers can collaborate on projects, as well as store and analyze data.

SaaS is the most common category of cloud computing. The SaaS provider manages everything from hardware stability to app functioning. Clients are not responsible for anything in this model; they only use programs to complete their tasks. In this case, the client software experience is fully dependent on the provider.

SaaS Key Features

- The subscription model of utilizing
- No need to download, install or upgrade software
- Resources can be scaled depending on requirements
- Apps are accessible from any connected device
- The provider is responsible for everything

SaaS Advantages

- No hardware costs
- No initial setup costs
- Automated upgrades
- Cross-device compatibility
- Accessible from any location
- Pay-as-you-go model
- Scalability
- Easy customization

SaaS Disadvantages

- Loss of control
- Limited range of solutions
- Connectivity is a must

When to Use SaaS

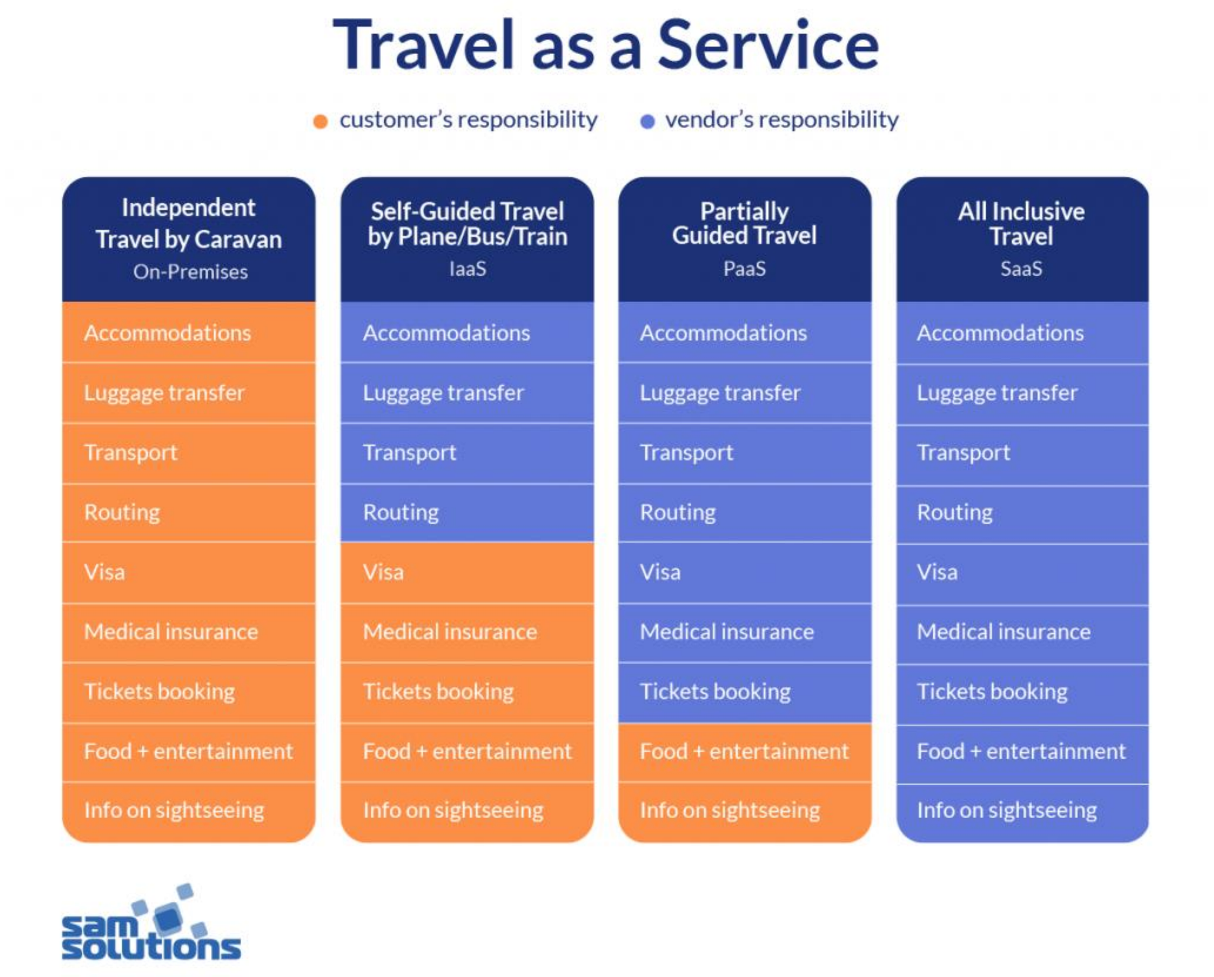
Utilizing SaaS is most beneficial in the following situations:

- If your company needs to launch a ready-made software quickly
- For short-term projects that require collaboration
- If you use applications on a temporary basis
- For applications that need both web and mobile access

Examples of SaaS

The best-known SaaS solutions vendors are Google Apps, Dropbox, Gmail, Salesforce, Cisco WebEx, Concur, GoToMeeting, Office365.

We’ve created a funny visual aid that establishes a parallel between different types of travel and different types of cloud services. We hope that this will help you better understand the difference between IaaS vs. PaaS vs. SaaS.

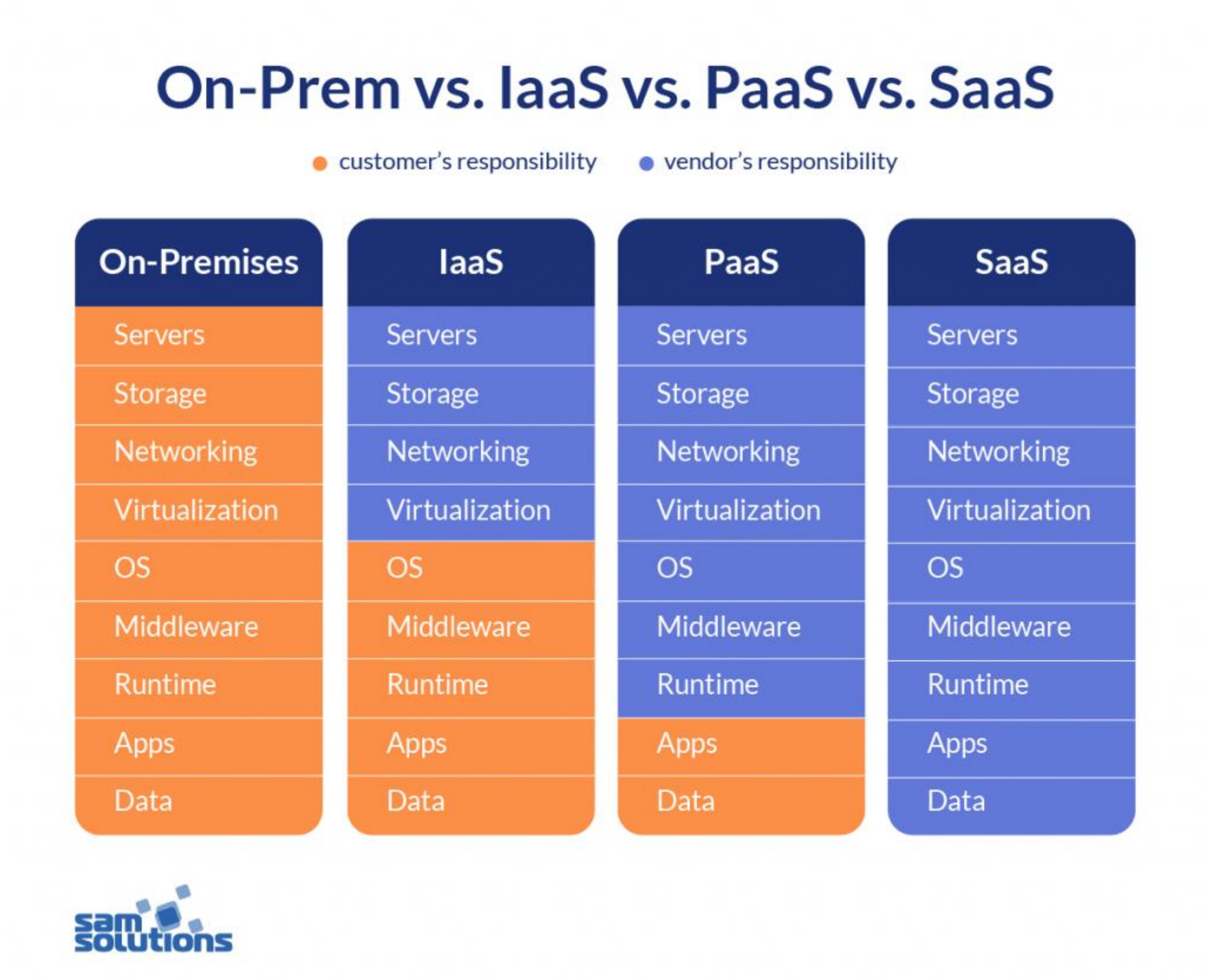


Travel as a Service

🟧 customer’s responsibility 🟦 vendor’s responsibility

Independent Travel by Caravan On-Premises	Self-Guided Travel by Plane/Bus/Train IaaS	Partially Guided Travel PaaS	All Inclusive Travel SaaS
🟧 Accommodations	🟦 Accommodations	🟦 Accommodations	🟦 Accommodations
🟧 Luggage transfer	🟦 Luggage transfer	🟦 Luggage transfer	🟦 Luggage transfer
🟧 Transport	🟦 Transport	🟦 Transport	🟦 Transport
🟧 Routing	🟦 Routing	🟦 Routing	🟦 Routing
🟧 Visa	🟧 Visa	🟦 Visa	🟦 Visa
🟧 Medical insurance	🟧 Medical insurance	🟦 Medical insurance	🟦 Medical insurance
🟧 Tickets booking	🟧 Tickets booking	🟦 Tickets booking	🟦 Tickets booking
🟧 Food + entertainment	🟧 Food + entertainment	🟧 Food + entertainment	🟦 Food + entertainment
🟧 Info on sightseeing	🟧 Info on sightseeing	🟧 Info on sightseeing	🟦 Info on sightseeing

sam solutions



On-Prem vs. IaaS vs. PaaS vs. SaaS

🟧 customer's responsibility 🟦 vendor's responsibility

On-Premises	IaaS	PaaS	SaaS
🟧 Servers	🟦 Servers	🟦 Servers	🟦 Servers
🟧 Storage	🟦 Storage	🟦 Storage	🟦 Storage
🟧 Networking	🟦 Networking	🟦 Networking	🟦 Networking
🟧 Virtualization	🟦 Virtualization	🟦 Virtualization	🟦 Virtualization
🟧 OS	🟧 OS	🟦 OS	🟦 OS
🟧 Middleware	🟧 Middleware	🟦 Middleware	🟦 Middleware
🟧 Runtime	🟧 Runtime	🟦 Runtime	🟦 Runtime
🟧 Apps	🟧 Apps	🟧 Apps	🟦 Apps
🟧 Data	🟧 Data	🟧 Data	🟦 Data

On-Premises/Independent travel

+ complete freedom

– you have to organize and manage everything by yourself

IaaS/Self-guided travel

+ basic amenities are provided by professionals

– you need to spend enough time on organization and management

PaaS/Partially guided travel

+ almost everything is organized by professionals, at the same time you have some freedom of action

– great dependency on the vendor

SaaS/All-inclusive travel

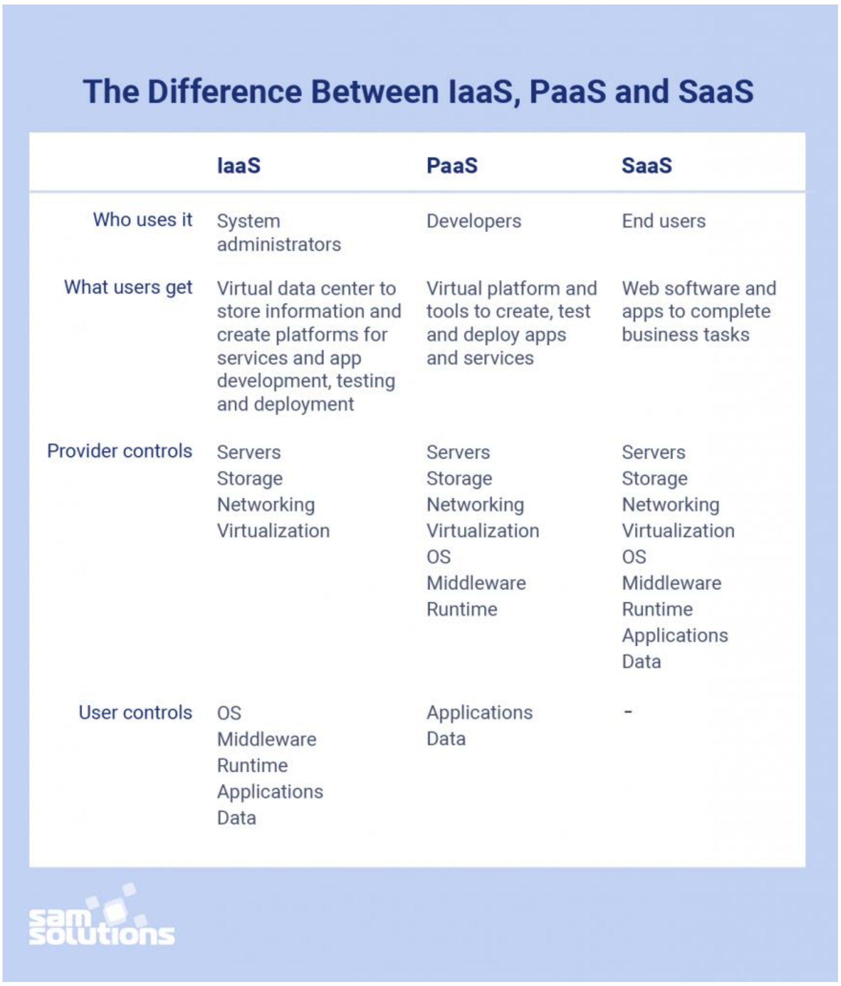
+ everything is organized for you by professionals

– no freedom, you fully depend on the vendor

Now, as you've got a rough idea of cloud service models, let's move on to their detailed examination.

The Difference Between IaaS, PaaS and SaaS

The table below provides a clear comparison of IaaS vs. PaaS vs. SaaS. Platform as a Service vs. Infrastructure as a Service gives less control to the user, but Platform as a Service vs. Software as a Service gives more control to the user. If you were to compare IaaS vs. SaaS, IaaS is the place you can move to and work from using available resources, while SaaS is a ready-made product you can utilize immediately without additional efforts.



The Difference Between IaaS, PaaS and SaaS

	IaaS	PaaS	SaaS
Who uses it	System administrators	Developers	End users
What users get	Virtual data center to store information and create platforms for services and app development, testing and deployment	Virtual platform and tools to create, test and deploy apps and services	Web software and apps to complete business tasks
Provider controls	Servers Storage Networking Virtualization	Servers Storage Networking Virtualization OS Middleware Runtime	Servers Storage Networking Virtualization OS Middleware Runtime Applications Data
User controls	OS Middleware Runtime Applications Data	Applications Data	-

sam solutions

OpenMPI Scheduling

static schedule means that iterations blocks are mapped statically to the execution threads in a round-robin fashion. The nice thing with static scheduling is that OpenMP run-time guarantees that if you have two separate loops with the same number of iterations and execute them with the same number of threads using static scheduling, then each thread will receive exactly the same iteration range(s) in both parallel regions. This is quite important on NUMA systems: if you touch some memory in the first loop, it will reside on the NUMA node where the executing thread was. Then in the second loop the same thread could access the same memory location faster since it will reside on the same NUMA node.

`dynamic` scheduling works on a "first come, first served" basis. Two runs with the same number of threads might (and most likely would) produce completely different "iteration space" -> "threads" mappings as one can easily verify:

There is another reason to choose between `static` and `dynamic` scheduling - workload balancing. If each iteration takes vastly different from the mean time to be completed then high work imbalance might occur in the static case. Take as an example the case where time to complete an iteration grows linearly with the iteration number. If iteration space is divided statically between two threads the second one will have three times more work than the first one and hence for 2/3 of the compute time the first thread will be idle. Dynamic schedule introduces some additional overhead but in that particular case will lead to much better workload distribution. A special kind of `dynamic` scheduling is the `guided` where smaller and smaller iteration blocks are given to each task as the work progresses.

Since precompiled code could be run on various platforms it would be nice if the end user can control the scheduling. That's why OpenMP provides the special `schedule(runtime)` clause. With `runtime` scheduling the type is taken from the content of the environment variable `OMP_SCHEDULE`. This allows to test different scheduling types without recompiling the application and also allows the end user to fine-tune for his or her platform.