ASYNCHRONOUS/SYNCHRONOUS COMPUTATION/COMMUNICATION

INTRODUCTION

Synchronous execution means the first task in a program must finish processing before moving on to executing the next task whereas Asynchronous execution means a second task can begin executing in parallel, without waiting for an earlier task to finish.

WHAT IS ASYNCHRONOUS?

Asynchronous communication happens when information can be exchanged independently of time. It doesn't require the recipient's immediate attention, allowing them to respond to the message at their convenience. Examples of asynchronous communication are emails, online forums, and collaborative documents.

EXAMPLES OF ASYNCHRONOUS COMMUNICATION:

- Email
- Letters or direct mail
- Project management tools
- Text messaging
- Direct messaging
- Video recordings (i.e., via Loom, Cloud-App, Drift Video, etc)
- Notes and action items in a Shared stream

THE BENEFITS OF AN ASYNCHRONOUS WORKPLACE

One of the main benefits of asynchronous work is that employees are able to work fully independently. So, logging onto a meeting at a specific time is a hindrance to their productivity and ability to work.

WHAT IS SYNCHRONOUS?

Synchronous communication happens in real-time, where at least two individuals are exchanging information, at the same time with each other. That's not to say that you need to be communicating in person for this to be deemed synchronous communication. This type of communication can be virtual as well, either scheduled or a little more impromptu.

SYNCHRONIZATION

- Managing the sequence of work and the tasks performing it is a critical design consideration for most parallel programs.
- Can be a significant factor in program performance (or lack of it)
- Often requires "serialization" of segments of the program.

EXAMPLES OF SYNCHRONOUS COMMUNICATION

- In-person meeting
- Phone call
- Video conference (i.e., via Zoom, WebEx, Slack, etc)

- Asking the teammate across your desk a quick question

THE BENEFITS OF A SYNCHRONOUS WORKPLACE

It's fair to say, communication that takes place in real-time lends itself to building more authentic connections, more so than the fragmented back-and-forth of asynchronous communications.

TYPES OF SYNCHRONIZATION

BARRIER

- Usually implies that all tasks are involved
- Each task performs its work until it reaches the barrier. It then stops, or "blocks".
- When the last task reaches the barrier, all tasks are synchronized.
- What happens from here varies. Often, a serial section of work must be done. In other cases, the tasks are automatically released to continue their work.

LOCK / SEMAPHORE

- Can involve any number of tasks
- Typically used to serialize (protect) access to global data or a section of code. Only one task at a time may use (own) the lock / semaphore / flag.
- The first task to acquire the lock "sets" it. This task can then safely (serially) access the protected data or code.
- Other tasks can attempt to acquire the lock but must wait until the task that owns the lock releases it.
- Can be blocking or non-blocking.

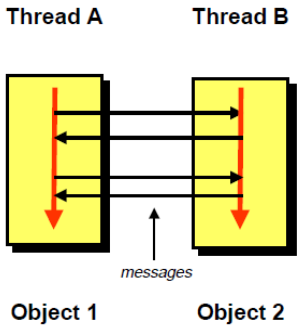
SYNCHRONOUS COMMUNICATION OPERATIONS

- Involves only those tasks executing a communication operation.
- When a task performs a communication operation, some form of coordination is required with the other task(s) participating in the communication. For example, before a task can perform a send operation, it must first receive an acknowledgment from the receiving task that it is OK to send.
- Discussed previously in the Communications section

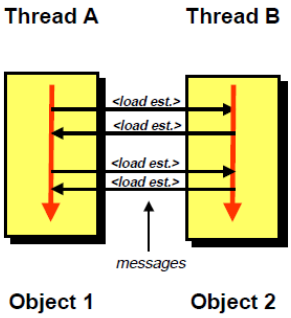
DIFFERENCE BETWEEN SYNCHRONOUS & ASYNCHRONOUS

SYNCHRONOUS	ASYNCHRONOUS
In synchronous communications, conversations happen instantaneously rather than the fragmented conversational snippets of email or online chat.	**Asynchronous communication** means participants of discourse don't have to respond immediately.
Synchronous communications might take the form of a video meeting or a phone call to a colleague on a mobile device.	Asynchronous transmission might take place via communication channels such as RingCentral messages, email exchanges, or messaging via a project management tool such as Trello or Asana.
In synchronous communications, participants are present and are expected to respond almost instantly	In asynchronous communications, they have little time to address their correspondence and craft a response.

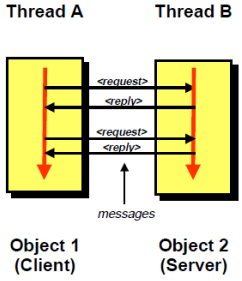
COMMUNICATING THREADS IN A MESSAGE-PASSING SYSTEM



MESSAGES MIGHT BE ASYNCHRONOUS IN NATURE

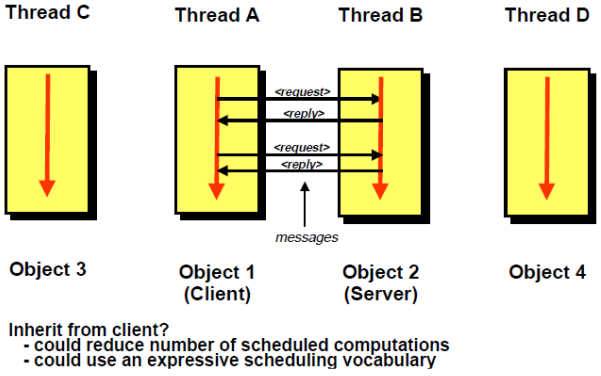


MESSAGES MIGHT BE SYNCHRONOUS IN NATURE



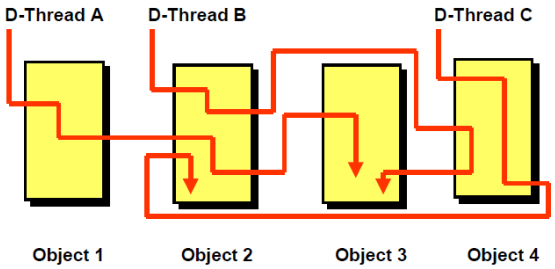
AN IMPORTANT SPECIAL CASE

SYNCHRONOUS CASE: HOW TO SCHEDULE THE SERVER'S WORK?



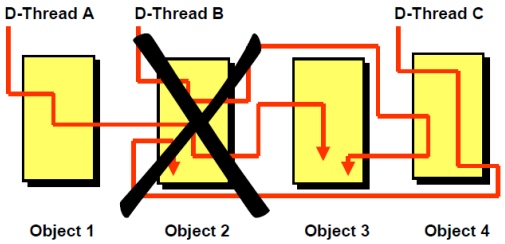
DISTRIBUTED (TRANS-NODE) THREADS:

SYNCHRONOUS INTERACTIONS (E.G., RPCS)



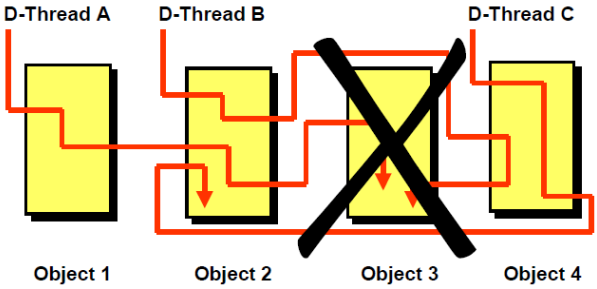
Bind Scheduling Information to Distributed Threads

DISTRIBUTED THREADS: PARTIAL FAILURES



ORPHANS, THREAD REPAIR, Etc.

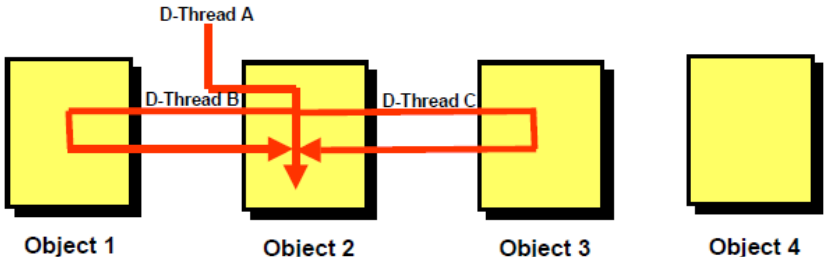
DISTRIBUTED THREADS:SUPPORT FOR ATOMIC (TRANS)ACTIONS



Thread Integrity Extends to All Visited Objects

WHAT ABOUT ASYNCHRONOUS COMPUTATIONS?

- Case I: independent computations
  - covered by distributed thread abstraction
- Case II: interacting computations
  - can utilize distributed threads
  - require higher-level support (e.g., distributed thread groups)



- **Message passing can be implemented on a synchronous model**

- Message passing can be controlled by manipulation of scheduling Parameters (e.g., declaration of message-passing time constraint)
- There is no particular support for scheduling the receiver's computation

- **what scheduling support would be useful?**
- **Are there common idioms or patterns that are valuable?**

CONCURRENCY CONTROL

In a database management system (DBMS), concurrency control manages simultaneous access to a database. It prevents two users from editing the same record at the same time and also serializes transactions for backup and recovery.

Concurrency control in DBMS is an important concept that is related to the transactions and data consistency of database management systems. Concurrency control refers to the process of managing independent operations of the database that are simultaneous and considered as a transaction in DBMS.

WHY IS CONCURRENCY CONTROL IMPORTANT?

Concurrency control is important to ensure data integrity when updates occur to the database in a multi-user environment. Concurrency control is used to apply isolation through mutual exclusion. It ensures serialization in the system. It preserves data consistency and resolves the conflict during read-write operations.

HOW DOES CONCURRENCY CONTROL WORK IN DBMS?

Concurrency Control is the process to maintain the data where there are multiple resources or users accessing the data element and performing the database operations. There are several enterprise systems such as banking, ticket booking, and traffic light systems that use a shared database as part of the data store associated with concurrent transactions. There is a chance of conflict for these transactions and resulting in data inconsistency.

CONCURRENCY CONTROL PROTOCOLS

Concurrency control protocols are the techniques used to maintain data consistency, atomicity, and serializability. Following are some of the concurrency control protocols.

1. LOCK BASED

The lock-based protocol is the technique of applying lock conditions on the data element, which helps in restricting another resource to perform read and write operations until the lock is active. There are mainly two types of lock such as shared or read-only lock and exclusive lock.

2. VALIDATION BASED

The validation-based protocol is also known as an optimistic concurrency control technique. It involves the read phase, validation phase, and writes phase for concurrency control.

3. TIMESTAMP BASED

The timestamp-based protocol uses system time or logical count as a timestamp to serialize the execution of concurrent transactions which helps to maintain the order of the transactions.

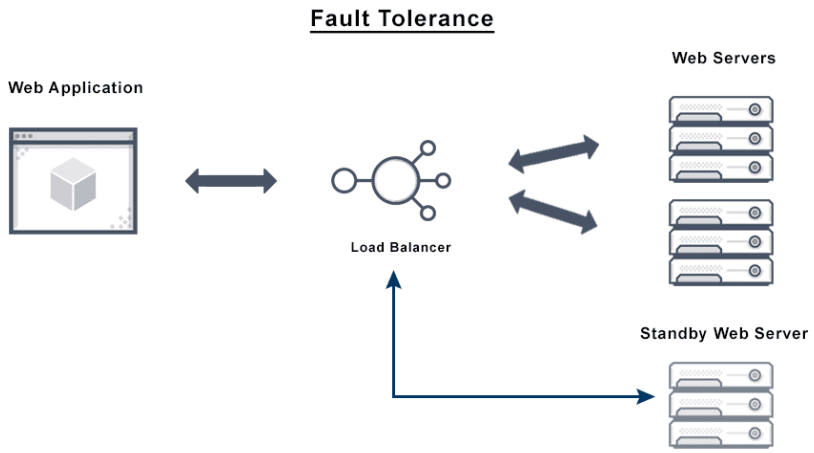
4. TWO-PHASE PROTOCOL

The two-phase protocol (2PL) is a locking mechanism that ensures serializability by using two distinct phases of lock condition. It uses the expanding phase and shrinking phase to acquire and release the lock condition to maintain concurrency control.

FAULT TOLERANCE

Fault tolerance refers to the ability of a system (computer, network, cloud cluster, etc.) to continue operating without interruption when one or more of its components fail.

The objective of creating a fault-tolerant system is to prevent disruptions arising from a single point of failure, ensuring the high availability and business continuity of mission-critical applications or systems.



Fault-tolerant systems use backup components that automatically take the place of failed components, ensuring no loss of service. These include:

- **Hardware Systems** that are backed up by identical or equivalent systems. For example, a server can be made fault tolerant by using an identical server running in parallel, with all operations mirrored to the backup server.
- **Software Systems** that are backed up by other software instances. For example, a database with customer information can be continuously replicated to another machine. If the primary database goes down, operations can be automatically redirected to the second database.
- **Power Sources** that are made fault tolerant using alternative sources. For example, many organizations have power generators that can take over in case mainline electricity fails.

In a similar fashion, any system or component which is a single point of failure can be made fault-tolerant using redundancy.

Fault tolerance can play a role in a disaster recovery strategy. For example, fault-tolerant systems with backup components in the cloud can restore mission-critical systems quickly, even if a natural or human-induced disaster destroys on-premise IT infrastructure.

WHY IS FAULT TOLERANCE IMPORTANT?

Fault tolerance is necessary for systems that are used to protect people's safety (such as air traffic control hardware and software systems), and in systems in which security, data protection and integrity, and high-value transactions depend.

WHICH INDUSTRIES DEPEND ON SYSTEM FAULT TOLERANCE?

Fault tolerance refers not only to the consequence of having redundant equipment but also to the ground-up methodology computer makers use to engineer and design their systems for reliability. Fault tolerance is a required design specification for computer equipment used in online transaction processing systems, such as airline flight control and reservations systems. Fault-tolerant systems are also widely used in sectors such as distribution and logistics, electric power plants, heavy manufacturing, industrial control systems and retailing.

WHAT ARE FAULT TOLERANCE REQUIREMENTS?

Depending on the fault tolerance issues that your organization copes with, there may be different fault tolerance requirements for your system. That is because fault-tolerant software and fault-tolerant hardware solutions both offer very high levels of availability, but in different ways.

Fault-tolerant servers use a minimal amount of system overhead to achieve high availability with an optimal level of performance. Fault-tolerant software may be able to run on servers you already have in place that meet industry standards.

WHAT IS FAULT TOLERANCE ARCHITECTURE?

There is more than one way to create a fault-tolerant server platform and thus prevent data loss and eliminate unplanned downtime. Fault tolerance in computer architecture simply reflects the decisions administrators and engineers use to ensure a system persists even after a failure. This is why there are various types of fault tolerance tools to consider. At the drive controller level, a redundant array of inexpensive disks (RAID) is a common fault tolerance strategy that can be implemented. Other facility level forms of fault tolerance exist, including cold, hot, warm, and mirror sites.

Fault tolerance computing also deals with outages and disasters. For this reason a fault tolerance strategy may include some uninterruptible power supply (UPS) such as a generator—some way to run independently from the grid should it fail. Byzantine fault tolerance (BFT) is another issue for modern fault tolerant architecture. BFT systems are important to the aviation, blockchain, nuclear power, and space industries because these systems prevent downtime even if certain nodes in a system fail or are driven by malicious actors.

WHAT IS THE RELATIONSHIP BETWEEN SECURITY AND FAULT TOLERANCE?

Fault tolerant design prevents security breaches by keeping your systems online and by ensuring they are well-designed. A naively-designed system can be taken offline easily by an attack, causing your organization to lose data, business, and trust. Each firewall, for example, that is not fault tolerant is a security risk for your site and organization.

WHAT IS FAULT TOLERANCE IN CLOUD COMPUTING?

Conceptually, fault tolerance in cloud computing is mostly the same as it is in hosted environments. Cloud fault tolerance simply means your infrastructure is capable of supporting uninterrupted functionality of your applications despite failures of components.

In a cloud computing setting that may be due to autoscaling across geographic zones or in the same data centers. There is likely more than one way to achieve fault-tolerant applications in the cloud in most cases. The overall system will still demand to monitor of available resources and potential failures, as with any fault tolerance in distributed systems.

WHAT ARE THE CHARACTERISTICS OF A FAULT TOLERANT DATA CENTER?

To be called a fault tolerant data center, a facility must avoid any single point of failure. Therefore, it should have two parallel systems for power and cooling. However, total duplication is costly, gains are not always worth that cost, and infrastructure is not the only answer. Therefore, many data centers practice fault avoidance strategies as a mid-level measure.

LOAD BALANCING FAULT TOLERANCE ISSUES

Load balancing and failover solutions can work together in the application delivery context. These strategies provide quicker recovery from disasters through redundancy, ensuring availability, which is why load balancing is part of many fault tolerant systems.

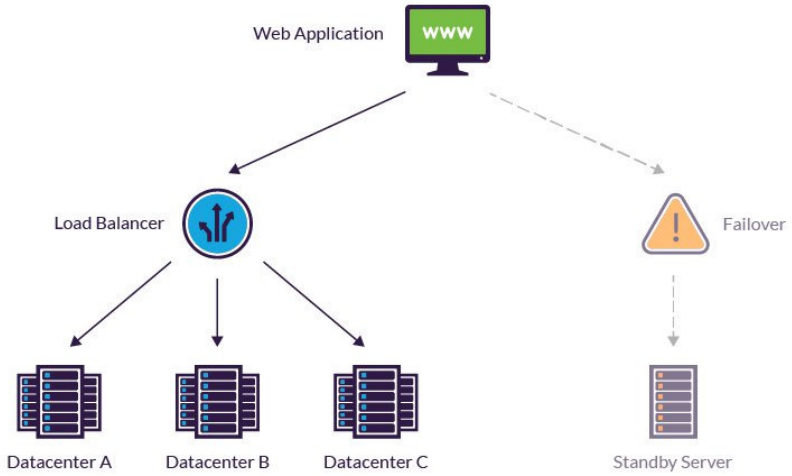
Load balancing solutions remove single points of failure, enabling applications to run on multiple network nodes. Most load balancers also make various computing resources more resilient to slowdowns and other disruptions by optimizing distribution of workloads across the system components. Load balancing also helps deal with partial network failures, shifting workloads when individual components experience problems.

DOES AVI NETWORKS OFFER A FAULT TOLERANCE SOLUTION?

Avi offers load balancing capabilities that can keep your systems online reliably. Aviaids fault tolerance by automatically instantiating virtual services when one fails, redistributing traffic, and handling workload moves or additions, reducing the chance of a single point of failure strangling your system.

WHERE IS FAULT TOLERANCE USED?

Fault tolerance can play a role in a disaster recovery strategy. For example, fault-tolerant systems with backup components in the cloud can restore mission-critical systems quickly, even if a natural or human-induced disaster destroys on-premise IT infrastructure.



WHAT IS AMDAHL'S LAW?

It is named after computer scientist Gene Amdahl (a computer architect from IBM and Amdahl corporation), and was presented at the AFIPS Spring Joint Computer Conference in 1967. It is also known as Amdahl's argument. It is a formula which gives the theoretical speedup in latency of the execution of a task at a fixed workload that can be expected of a system whose resources are improved. In other words, it is a formula used to find the maximum improvement possible by just improving a particular part of a system. It is often used in parallel computing to predict the theoretical speedup when using multiple processors.

SPEEDUP

Speedup is defined as the ratio of performance for the entire task using the enhancement and performance for the entire task without using the enhancement or speedup can be defined as the ratio of execution time for the entire task without using the enhancement and execution time for the entire task using the enhancement.

If Pe is the performance for entire task using the enhancement when possible, Pw is the performance for entire task without using the enhancement, Ew is the execution time for entire task without using the enhancement and Ee is the execution time for entire task using the enhancement when possible then,

Speedup = Pe/Pw

or

Speedup = Ew/Ee

Amdahl's law uses two factors to find speedup from some enhancement –

- **Fraction Enhanced** – The fraction of the computation time in the original computer that can be converted to take advantage of the enhancement. For example- if 10 seconds of the execution time of a program that takes 40 seconds in total can use an enhancement, the fraction is 10/40. This obtained value is Fraction Enhanced.

Fraction enhanced is always less than 1.

- **Speedup Enhanced** – The improvement gained by the enhanced execution mode; that is, how much faster the task would run if the enhanced mode were used for the entire program. For example – If the enhanced mode takes, say 3 seconds for a portion of the program, while it is 6 seconds in the original mode, the improvement is 6/3. This value is Speedup enhanced.

Speedup Enhanced is always greater than 1.

THE OVERALL SPEEDUP IS THE RATIO OF THE EXECUTION TIME

$$\text{Overall Speedup} = \frac{\text{Old execution time}}{\text{New execution time}}$$

$$= \frac{1}{\left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)}$$

Amdahl's law is an expression used to find the maximum expected improvement to an overall system whe nonly part of the system is improved. It is often used in parallel computing to predict the theoretical maximum speedup using multiple processors.

Suppose, Moni have to attend an invitation. Moni's another two friend Diya and Hena are also invited. There are conditions that all three friends have to go there separately and all of them have to be present at door to get into the hall. Now Moni is coming by car, Diya by bus and Hena is coming by foot. Now, how fast Moni and Diya can reach there it doesn't matter, they have to wait for Hena. So to speed up the overall process, we need to concentrate on the performance of Hena other than Moni or Diya.

This is actually happening in Amdahl's Law. It relates the improvement of the system's performance with the parts that didn't perform well, like we need to take care of the performance of that parts of the system. This law often used in parallel computing to predict the theoretical speedup when using multiple processors.

WHAT KINDS OF PROBLEMS DO WE SOLVE WITH AMDAHL'S LAW?

Recall how we defined the performance of a system that has been sped up:

$$\text{Speedup} = \frac{\text{Execution Time Before Improvement}}{\text{Execution Time After Improvement}}$$

There are three types of problems to be solved using the following Amdahl's Law equation:

$$\text{Speedup} = \frac{1}{(1 - \text{fraction enhanced}) + (\text{fraction enhanced}/\text{factor of improvement})}$$

Let Speedup be denoted by "S", fraction enhanced be denoted by "fE", and factor of improvement bedenoted by "fI". Then we can write the above equation as

$$S = ((1 - fE) + (fE / fI))^{-1}$$

The three problem types are as follows:

1) Determine S given fE and fI
2) Determine fI given S and fE
3) Determine fE given S and fI

STATE AND PROVE AMDAHL'S LAW TO COMPUTE SPEEDUP OF PARALLEL COMPUTERS

Amdahl's law is related to the speedup of a parallel computer. When a program is run on a parallel computer then the computation may be serial, parallel or both. At times, there will be a certain part of the program which has to run serially. This part is termed to be a sequential fraction of computing. Consider the sequential fraction of the program to be F. Then the parallel computation of the program will be 1-F. Based on this, Amdahl's law states that the speedup of parallel computers is given by the relation.

$$S(p) \leq \frac{1}{F + \frac{1-F}{p}}$$

where 'p' is the number of processors. To prove the above relation, let us take an example of execution of a task on a computer with a single processor and p number of processors. We already know that the speedup is given by the relation,

$$S(p) = \frac{T(s)}{T(p)} \quad ..(1)$$

Consider that the execution time required to complete the given task on a computer with single processor is T. When the same task is executed on parallel computer with p processors, then the time required will depend upon sequential computing time and parallel computing time that is,

T(p) = Sequential computing Time + Parallel Computing Time

$$T(p) = (F).(T) + \frac{(1-F).T}{p}$$

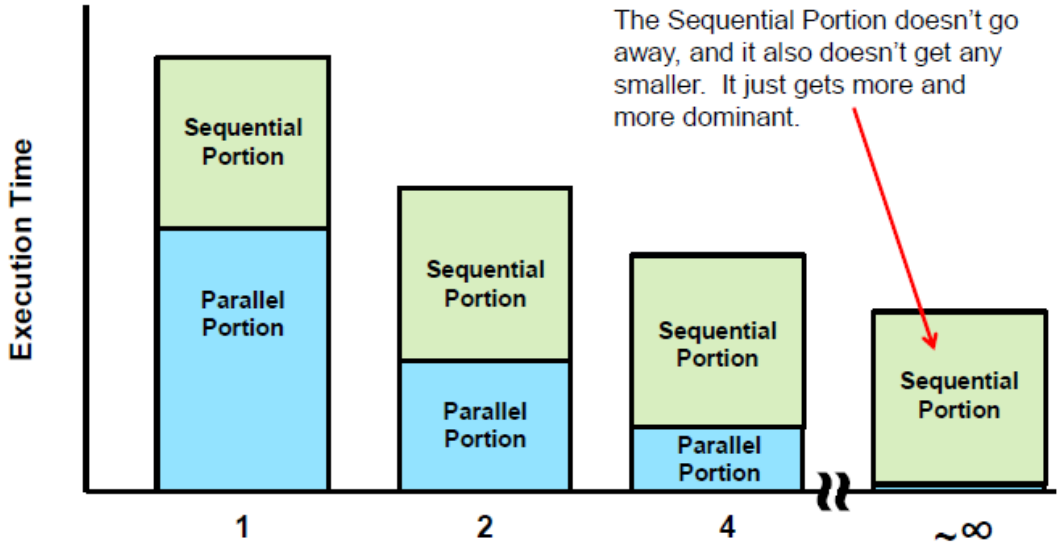
where 'p' is the number of processors. Therefore, by equation 1

$$S(p) = \frac{T}{(F).(T) + \frac{(1-F).T}{p}} = \frac{1}{F + \frac{1-F}{p}} \quad ...(2)$$

From equation (2), we can say that S(p) will always be ≤ 1, that is

$$S(p) \leq \frac{1}{F + \frac{1-F}{p}}$$

A Visual Explanation of Amdahl's Law



You can also solve for $F_{parallel}$ using Amdahl's Law if you know your speedup and the number of processors

Amdahl's law says:

$$S = \frac{T_1}{T_n} = \frac{1}{\frac{F}{n} + (1-F)} \Rightarrow \frac{1}{S} = \frac{F}{n} + (1-F) = 1 + \frac{F - nF}{n} \Rightarrow \frac{1}{S} - 1 = F\frac{(1-n)}{n}$$

Use this if you know the timing

Use this if you know the speedup

Solving for F:

$$F = \frac{\frac{1}{S} - 1}{\frac{1-n}{n}} = \frac{\frac{T_n}{T_1} - 1}{\frac{1-n}{n}} = \frac{\frac{T_n - T_1}{T_1}}{\frac{1-n}{n}} = \frac{\frac{T_1 - T_n}{T_1}}{\frac{n-1}{n}} = \frac{n(T_1 - T_n)}{T_1(n-1)} = \frac{n}{(n-1)}\frac{T_1 - T_n}{T_1} = \frac{n}{(n-1)}\left(1 - \frac{1}{Speedup}\right)$$

If you've got several (n,S) values, you can take the average (which is actually a least squares fit):

$$F_i = \frac{n_i}{(n_i - 1)}\frac{T_1 - T_{n_i}}{T_1}, i = 2..N$$

$$\overline{F} = \frac{\sum_{i=2}^{N} F_i}{N-1}$$

note that when i=1, $T_{ni} = T_1$

Amdahl's Law can also give us the Maximum Possible SpeedUp

Note that these fractions put an upper bound on how much benefit you will get from adding more processors:

$$\max Speedup = \lim_{n\to\infty} Speedup = \frac{1}{F_{sequential}} = \frac{1}{1 - F_{parallel}}$$

Fparallel	maxSpeedup
0.00	1.00
0.10	1.11
0.20	1.25
0.30	1.43
0.40	1.67
0.50	2.00
0.60	2.50
0.70	3.33
0.80	5.00
0.90	10.00
0.95	20.00
0.99	100.00

A More Optimistic Take on Amdahl's Law: The Gustafson-Baris Observation

Gustafson observed that as you increase the number of processors, you have a tendency to attack larger and larger versions of the problem. He also observed that when you use the same parallel program on larger datasets, the parallel fraction, F_p, increases.

Let P be the amount of time spent on the parallel portion of an original task and S spent on the serial portion. Then

$$F_p = \frac{P}{P + S}$$

Parallel Time

Serial Time

or

$$S = \frac{P - PF_p}{F_p}$$

Without loss of generality, we can set P=1 so that, really, S is now a fraction of P. We now have:

$$S = \frac{1 - F_p}{F_p}$$

A More Optimistic Take on Amdahl's Law: The Gustafson-Baris Observation

We know that if we multiply the amount of data to process by *N*, then the amount of parallel work becomes *NP*. Surely the serial work must increase too, but we don't know how much. Let's say it doesn't increase at all, so that we know we are getting an upper bound answer.

In that case, the new parallel fraction is: $F_p^{'} = \frac{P'}{P'+S} = \frac{NP}{NP+S}$

And substituting for *P* (=1) and for *S*, we have:

$$F_p^{'} = \frac{N}{N+S} = \frac{N}{N+\frac{1-F_p}{F_p}}$$

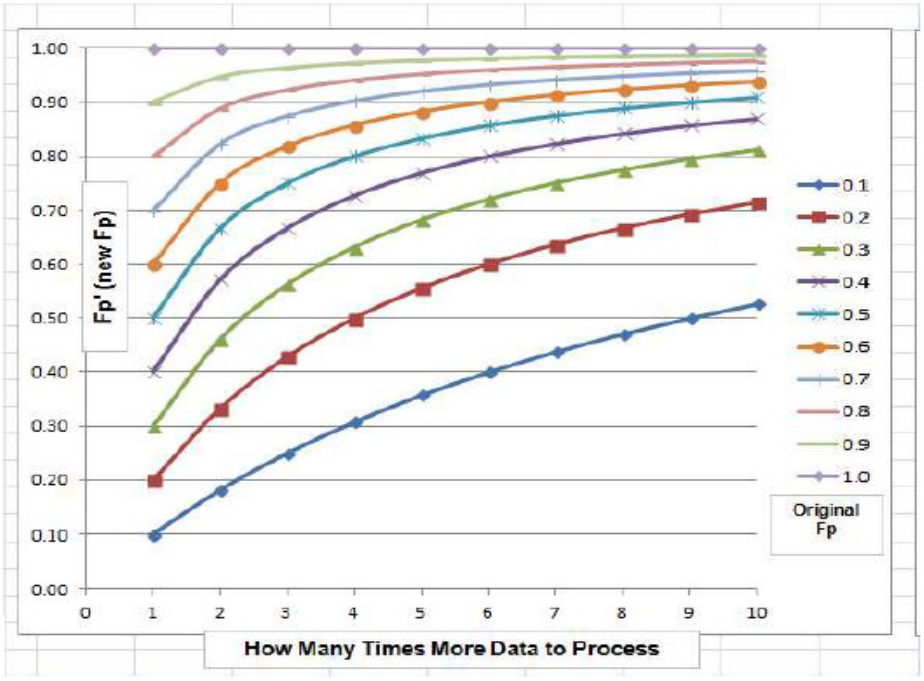
A More Optimistic Take on Amdahl's Law: The Gustafson-Baris Observation

If we tabulate this, we get a table of F_p' values:

Original Fp	How Many Times More Data to Process									
	1	2	3	4	5	6	7	8	9	10
0.1	0.10	0.18	0.25	0.31	0.36	0.40	0.44	0.47	0.50	0.53
0.2	0.20	0.33	0.43	0.50	0.56	0.60	0.64	0.67	0.69	0.71
0.3	0.30	0.46	0.56	0.63	0.68	0.72	0.75	0.77	0.79	0.81
0.4	0.40	0.57	0.67	0.73	0.77	0.80	0.82	0.84	0.86	0.87
0.5	0.50	0.67	0.75	0.80	0.83	0.86	0.88	0.89	0.90	0.91
0.6	0.60	0.75	0.82	0.86	0.88	0.90	0.91	0.92	0.93	0.94
0.7	0.70	0.82	0.88	0.90	0.92	0.93	0.94	0.95	0.95	0.96
0.8	0.80	0.89	0.92	0.94	0.95	0.96	0.97	0.97	0.97	0.98
0.9	0.90	0.95	0.96	0.97	0.98	0.98	0.98	0.99	0.99	0.99
1.0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

A More Optimistic Take on Amdahl's Law: The Gustafson-Baris Observation

Or, graphing it:



A More Optimistic Take on Amdahl's Law: The Gustafson-Baris Observation

We can also turn F_p' into a Maximum Speedup: