**SCHEDULING**

In [computing](https://en.wikipedia.org/wiki/Computing), **scheduling** is the method by which work specified by some means is assigned to resources that complete the work. The work may be virtual computation elements such as [threads](https://en.wikipedia.org/wiki/Thread_(computer_science)), [processes](https://en.wikipedia.org/wiki/Process_(computing)) or data [flows](https://en.wikipedia.org/wiki/Flow_(computer_networking)), which are in turn scheduled onto hardware resources such as [processors](https://en.wikipedia.org/wiki/Central_processing_unit), network links or [expansion cards](https://en.wikipedia.org/wiki/Expansion_card).

A scheduler is what carries out the scheduling activity. Schedulers are often implemented so they keep all computer resources busy (as in [load balancing](https://en.wikipedia.org/wiki/Load_balancing_(computing))), allow multiple users to share system resources effectively, or to achieve a target [quality of service](https://en.wikipedia.org/wiki/Quality_of_service). Scheduling is fundamental to computation itself, and an intrinsic part of the [execution model](https://en.wikipedia.org/wiki/Execution_model) of a computer system; the concept of scheduling makes it possible to have [computer multitasking](https://en.wikipedia.org/wiki/Computer_multitasking) with a single [central processing unit](https://en.wikipedia.org/wiki/Central_processing_unit) (CPU).

A scheduler may aim at one of many goals, for example, maximizing [*throughput*](https://en.wikipedia.org/wiki/Throughput) (the total amount of work completed per time unit), minimizing [*response time*](https://en.wikipedia.org/wiki/Responsiveness) (time from work becoming enabled until the first point it begins execution on resources), or minimizing [*latency*](https://en.wikipedia.org/wiki/Latency_(engineering)) (the time between work becoming enabled and its subsequent completion), maximizing *fairness* (equal CPU time to each process, or more generally appropriate times according to the priority and workload of each process). In practice, these goals often conflict (e.g. throughput versus latency), thus a scheduler will implement a suitable compromise. Preference is given to any one of the concerns mentioned above, depending upon the user's needs and objectives.

In [real-time](https://en.wikipedia.org/wiki/Real-time_computing) environments, such as [embedded systems](https://en.wikipedia.org/wiki/Embedded_system) for [automatic control](https://en.wikipedia.org/wiki/Automatic_control) in industry (for example [robotics](https://en.wikipedia.org/wiki/Robotics)), the scheduler also must ensure that processes can meet [deadlines](https://en.wikipedia.org/wiki/Time_limit); this is crucial for keeping the system stable. Scheduled tasks can also be distributed to remote devices across a network and [managed](https://en.wikipedia.org/wiki/Device_Management) through an administrative back end.

**PERFORMANCE**

**performance** is the amount of work accomplished by a computer system. Depending on the context, high computer performance may involve one or more of the following:

* Short [response time](https://en.wikipedia.org/wiki/Response_time_(technology)) for a given piece of work
* High [throughput](https://en.wikipedia.org/wiki/Throughput) (rate of processing work)
* Low utilization of [computing resource](https://en.wikipedia.org/wiki/Computing_resource)(s)
* [High availability](https://en.wikipedia.org/wiki/High_availability) of the computing system or application
* Fast (or highly compact) [data compression](https://en.wikipedia.org/wiki/Data_compression) and decompression
* High [bandwidth](https://en.wikipedia.org/wiki/Bandwidth_(computing))
* Short [data transmission](https://en.wikipedia.org/wiki/Data_transmission) time

RELIABILITY  
*Reliability* can be defined as the probability that a system will produce correct outputs up to some given time *t*.[[5]](https://en.wikipedia.org/wiki/Reliability,_availability_and_serviceability#cite_note-McClusky-5) Reliability is enhanced by features that help to avoid, detect and repair hardware faults. A reliable system does not silently continue and deliver results that include uncorrected corrupted data. Instead, it detects and, if possible, corrects the corruption, for example: by retrying an operation for transient ([soft](https://en.wikipedia.org/wiki/Soft_error)) or intermittent errors, or else, for uncorrectable errors, isolating the fault and reporting it to higher-level recovery mechanisms (which may [failover](https://en.wikipedia.org/wiki/Failover) to redundant replacement hardware, etc.), or else by halting the affected program or the entire system and reporting the corruption. Reliability can be characterized in terms of [mean time between failures](https://en.wikipedia.org/wiki/Mean_time_between_failures) (MTBF), with reliability = exp(-t/MTBF).

FAULT RECOVERY

**Fault tolerance** is the property that enables a [system](https://en.wikipedia.org/wiki/System) to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naively designed system in which even a small failure can cause total breakdown. Fault tolerance is particularly sought after in [high-availability](https://en.wikipedia.org/wiki/High-availability) or [life-critical systems](https://en.wikipedia.org/wiki/Life-critical_system). The ability of maintaining functionality when portions of a system break down is referred to as **graceful degradation**.[[1]](https://en.wikipedia.org/wiki/Fault_tolerance#cite_note-1)

A **fault-tolerant design** enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system [fails](https://en.wikipedia.org/wiki/Failure).[[2]](https://en.wikipedia.org/wiki/Fault_tolerance#cite_note-2) The term is most commonly used to describe [computer systems](https://en.wikipedia.org/wiki/Computer_system) designed to continue more or less fully operational with, perhaps, a reduction in [throughput](https://en.wikipedia.org/wiki/Throughput) or an increase in [response time](https://en.wikipedia.org/wiki/Response_time_(technology)) in the event of some partial failure. That is, the system as a whole is not stopped due to problems either in the [hardware](https://en.wikipedia.org/wiki/Computer_hardware) or the [software](https://en.wikipedia.org/wiki/Software). An example in another field is a motor vehicle designed so it will continue to be drivable if one of the tires is punctured, or a structure that is able to retain its integrity in the presence of damage due to causes such as [fatigue](https://en.wikipedia.org/wiki/Fatigue_(material)), [corrosion](https://en.wikipedia.org/wiki/Corrosion), manufacturing flaws, or impact.

Within the scope of an *individual* system, fault tolerance can be achieved by anticipating exceptional conditions and building the system to cope with them, and, in general, aiming for [self-stabilization](https://en.wikipedia.org/wiki/Self-stabilization) so that the system converges towards an error-free state. However, if the consequences of a system failure are catastrophic, or the cost of making it sufficiently reliable is very high, a better solution may be to use some form of duplication. In any case, if the consequence of a system failure is so catastrophic, the system must be able to use reversion to fall back to a safe mode. This is similar to roll-back recovery but can be a human action if humans are present in the loop.

A highly fault-tolerant system might continue at the same level of performance even though one or more components have failed. For example, a building with a backup electrical generator will provide the same voltage to wall outlets even if the grid power fails.

A system that is designed to [fail safe](https://en.wikipedia.org/wiki/Fail_safe), or fail-secure, or **fail gracefully**, whether it functions at a reduced level or fails completely, does so in a way that protects people, property, or data from injury, damage, intrusion, or disclosure. In computers, a program might fail-safe by executing a [graceful exit](https://en.wikipedia.org/wiki/Graceful_exit) (as opposed to an uncontrolled crash) in order to prevent data corruption after experiencing an error. A similar distinction is made between "failing well" and "[failing badly](https://en.wikipedia.org/wiki/Failing_badly)".

[Fail-deadly](https://en.wikipedia.org/wiki/Fail-deadly) is the opposite strategy, which can be used in weapon systems that are designed to kill or injure targets even if part of the system is damaged or destroyed.

A system that is designed to experience **graceful degradation**, or to **fail soft** (used in computing, similar to "fail safe"[[3]](https://en.wikipedia.org/wiki/Fault_tolerance#cite_note-3)) operates at a reduced level of performance after some component failures. For example, a building may operate lighting at reduced levels and elevators at reduced speeds if grid power fails, rather than either trapping people in the dark completely or continuing to operate at full power. In computing an example of graceful degradation is that if insufficient network bandwidth is available to stream an online video, a lower-resolution version might be streamed in place of the high-resolution version. [Progressive enhancement](https://en.wikipedia.org/wiki/Progressive_enhancement) is an example in computing, where web pages are available in a basic functional format for older, small-screen, or limited-capability web browsers, but in an enhanced version for browsers capable of handling additional technologies or that have a larger display available.

In [fault-tolerant computer systems](https://en.wikipedia.org/wiki/Fault-tolerant_computer_system), programs that are considered [robust](https://en.wikipedia.org/wiki/Robustness_(computer_science)) are designed to continue operation despite an error, exception, or invalid input, instead of crashing completely. [Software brittleness](https://en.wikipedia.org/wiki/Software_brittleness) is the opposite of robustness. [Resilient networks](https://en.wikipedia.org/wiki/Resilience_(network)) continue to transmit data despite the failure of some links or nodes; [resilient buildings and infrastructure](https://en.wikipedia.org/wiki/Resilience_(engineering_and_construction)) are likewise expected to prevent complete failure in situations like earthquakes, floods, or collisions.

A system with high [failure transparency](https://en.wikipedia.org/wiki/Failure_transparency) will alert users that a component failure has occurred, even if it continues to operate with full performance, so that failure can be repaired or imminent complete failure anticipated. Likewise, a [fail-fast](https://en.wikipedia.org/wiki/Fail-fast) component is designed to report at the first point of failure, rather than allow downstream components to fail and generate reports then. This allows easier diagnosis of the underlying problem, and may prevent improper operation in a broken state.

DISTRIBUTED SYSTEMS

A distributed system is a network that consists of autonomous computers that are connected using a distribution middleware. They help in sharing different resources and capabilities to provide users with a single and integrated coherent network.

The key features of a distributed system are:

* Components in the system are concurrent. A distributed system allows resource sharing, including software by systems connected to the network at the same time.
* The components could be multiple but will generally be autonomous in nature.
* A global clock is not required in a distributed system. The systems can be spread across different geographies.
* Compared to other network models, there is greater fault tolerance in a distributed model.
* Price/performance ratio is much better.

The key goals of a distributed system include:

* Transparency: Achieving the image of a single system image without concealing the details of the location, access, migration, concurrency, failure, relocation, persistence and resources to the users.
* Openness: Making the network easier to configure and modify.
* Reliability: Compared to a single system, a distributed system should be highly capable of being secure, consistent and have a high capability of masking errors.
* Performance: Compared to other models, distributed models are expected to give a much-wanted boost to performance.
* Scalability: Distributed systems should be scalable with respect to geography, administration or size.

Challenges for the distributed system include:

* Security is a big challenge in a distributed environment, especially when using public networks.
* Fault tolerance could be tough when the distributed model is built based on unreliable components.
* Coordination and resource sharing can be difficult if proper protocols or policies are not in place.
* Process knowledge should be put in place for the administrators and users of the distributed model.

HETEROGENEOUS DISTRIBUTED SYSTEMS

**Heterogeneous computing** refers to systems that use more than one kind of processor or cores. These systems gain performance or energy efficiency not just by adding the same type of processors, but by adding dissimilar [coprocessors](https://en.wikipedia.org/wiki/Coprocessors), usually incorporating specialized processing capabilities to handle particular tasks.

Usually heterogeneity in the context of computing referred to different [instruction set architectures](https://en.wikipedia.org/wiki/Instruction_set_architecture) (ISA), where the main processor has one and the rest have another, usually a very different architecture (maybe more than one), not just a different [microarchitecture](https://en.wikipedia.org/wiki/Microarchitecture) ([floating point](https://en.wikipedia.org/wiki/Floating_point) number processing is a special case of this not usually referred to as heterogeneous). E.g. [ARM big.LITTLE](https://en.wikipedia.org/wiki/ARM_big.LITTLE) is an exception where the ISAs of cores are the same and heterogeneity refers to the speed of different microarchitectures of the same ISA,[[2]](https://en.wikipedia.org/wiki/Heterogeneous_computing" \l "cite_note-2) then making it more like a [symmetric multiprocessor](https://en.wikipedia.org/wiki/Symmetric_multiprocessor) (SMP).

In the past heterogeneous computing meant different ISAs had to be handled differently, while a modern example, [Heterogeneous System Architecture](https://en.wikipedia.org/wiki/Heterogeneous_System_Architecture) (HSA) systems,[[3]](https://en.wikipedia.org/wiki/Heterogeneous_computing#cite_note-hsa_foundation-3) eliminate the difference (for the user); use multiple processor types (typically [CPUs](https://en.wikipedia.org/wiki/CPU) and [GPUs](https://en.wikipedia.org/wiki/GPU)[[4]](https://en.wikipedia.org/wiki/Heterogeneous_computing#cite_note-4)), usually on the same [integrated circuit](https://en.wikipedia.org/wiki/Integrated_circuit), to provide the best of both worlds: general GPU processing (apart from its well-known 3D graphics rendering capabilities, it can also perform mathematically intensive computations on very large data sets), while CPUs can run the operating system and perform traditional serial tasks.

The level of heterogeneity in modern computing systems is gradually increasing as further scaling of fabrication technologies allows for formerly discrete components to become integrated parts of a [system-on-chip](https://en.wikipedia.org/wiki/System_on_a_chip), or SoC. For example, many new processors now include built-in logic for interfacing with other devices ([SATA](https://en.wikipedia.org/wiki/SATA), [PCI](https://en.wikipedia.org/wiki/Conventional_PCI), [Ethernet](https://en.wikipedia.org/wiki/Ethernet), [USB](https://en.wikipedia.org/wiki/USB), [RFID](https://en.wikipedia.org/wiki/RFID), [Radios](https://en.wikipedia.org/wiki/Radio), [UARTs](https://en.wikipedia.org/wiki/UART), and [memory controllers](https://en.wikipedia.org/wiki/Memory_controller)), as well as programmable functional units and [hardware accelerators](https://en.wikipedia.org/wiki/Hardware_acceleration) ([GPUs](https://en.wikipedia.org/wiki/GPU), [cryptography](https://en.wikipedia.org/wiki/Cryptography) [co-processors](https://en.wikipedia.org/wiki/Coprocessor), programmable network processors, A/V encoders/decoders, etc.).

A computational task is typically *replicated in space*, i.e. executed on separate devices, or it could be *replicated in time*, if it is executed repeatedly on a single device. Replication in space or in time is often linked to scheduling algorithms [[1]](https://en.wikipedia.org/wiki/Replication_(computing)#cite_note-1)

The access to a replicated entity is typically uniform with access to a single, non-replicated entity. The replication itself should be [transparent](https://en.wikipedia.org/wiki/Transparency_(human-computer_interaction)) to an external user. Also, in a failure scenario, a [failover](https://en.wikipedia.org/wiki/Failover) of replicas is hidden as much as possible. The latter refers to data replication with respect to [Quality of Service (QoS)](https://en.wikipedia.org/wiki/Quality_of_service) aspects.[[2]](https://en.wikipedia.org/wiki/Replication_(computing)#cite_note-2)

Computer scientists talk about active and passive replication in systems that replicate data or services:

* *active replication* is performed by processing the same request at every replica.
* *passive replication* involves processing each single request on a single replica and then transferring its resultant state to the other replicas.

If at any time one master replica is designated to process all the requests, then we are talking about the *primary-backup* scheme ([*master-slave*](https://en.wikipedia.org/wiki/Master-slave_(computers)) scheme) predominant in [high-availability clusters](https://en.wikipedia.org/wiki/High-availability_cluster). On the other side, if any replica processes a request and then distributes a new state, then this is a *multi-primary* scheme (called [*multi-master*](https://en.wikipedia.org/wiki/Multi-master_replication) in the database field). In the multi-primary scheme, some form of [distributed concurrency control](https://en.wikipedia.org/wiki/Distributed_concurrency_control) must be used, such as [distributed lock manager](https://en.wikipedia.org/wiki/Distributed_lock_manager).

[Load balancing](https://en.wikipedia.org/wiki/Load_balancing_(computing)) differs from task replication, since it distributes a load of different (not the same) computations across machines, and allows a single computation to be dropped in case of failure. Load balancing, however, sometimes uses data replication (especially multi-master replication) internally, to distribute its data among machines.

[Backup](https://en.wikipedia.org/wiki/Backup) differs from replication in that it saves a copy of data unchanged for a long period of time.[*[citation needed](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed" \o "Wikipedia:Citation needed)*] Replicas, on the other hand, undergo frequent updates and quickly lose any historical state. Replication is one of the oldest and most important topics in the overall area of [distributed systems](https://en.wikipedia.org/wiki/Distributed_computing).

Whether one replicates data or computation, the objective is to have some group of processes that handle incoming events. If we replicate data, these processes are passive and operate only to maintain the stored data, reply to read requests, and apply updates. When we replicate computation, the usual goal is to provide fault-tolerance. For example, a replicated service might be used to control a telephone switch, with the objective of ensuring that even if the primary controller fails, the backup can take over its functions. But the underlying needs are the same in both cases: by ensuring that the replicas see the same events in equivalent orders, they stay in consistent states and hence any replica can respond to queries.

TASK  
In computer programming, a task is a basic unit of programming that an [operating system](http://searchcio-midmarket.techtarget.com/definition/operating-system) controls. Depending on how the operating system defines a task in its design, this unit of programming may be an entire [program](http://searchsoftwarequality.techtarget.com/definition/program) or each successive invocation of a program. Since one program may make requests of other [utility](http://searchcio-midmarket.techtarget.com/definition/utility) programs, the utility programs may also be considered tasks (or subtasks). All of today's widely-used operating systems support [multitasking](http://searchcio-midmarket.techtarget.com/definition/multitasking) , which allows multiple tasks to run concurrently, taking turns using the resources of the computer.

PROCESSOR  
A processor, or "microprocessor," is a small chip that resides in [computers](https://techterms.com/definition/computer) and other electronic devices. Its basic job is to receive [input](https://techterms.com/definition/input) and provide the appropriate [output](https://techterms.com/definition/output). While this may seem like a simple task, modern processors can handle trillions of calculations per second.

The central processor of a computer is also known as the [CPU](https://techterms.com/definition/cpu), or "central processing unit." This processor handles all the basic system instructions, such as processing [mouse](https://techterms.com/definition/mouse) and [keyboard](https://techterms.com/definition/keyboard) input and running [applications](https://techterms.com/definition/application). Most [desktop computers](https://techterms.com/definition/desktop_computer) contain a CPU developed by either Intel or AMD, both of which use the [x86](https://techterms.com/definition/x86) processor [architecture](https://techterms.com/definition/architecture). Mobile devices, such as [laptops](https://techterms.com/definition/laptop) and [tablets](https://techterms.com/definition/tablet) may use Intel and AMD CPUs, but can also use specific mobile processors developed by companies like ARM or Apple.

Modern CPUs often include multiple processing cores, which work together to process instructions. While these "cores" are contained in one physical unit, they are actually individual processors. In fact, if you view your computer's performance with a system monitoring [utility](https://techterms.com/definition/utility) like Windows Task Manager (Windows) or Activity Monitor (Mac OS X), you will see separate graphs for each processor. Processors that include two cores are called [dual-core](https://techterms.com/definition/dualcore) processors, while those with four cores are called [quad-core](https://techterms.com/definition/quadcore) processors. Some high-end workstations contain multiple CPUs with multiple cores, allowing a single machine to have eight, twelve, or even more processing cores.

Besides the central processing unit, most desktop and laptop computers also include a [GPU](https://techterms.com/definition/gpu). This processor is specifically designed for rendering graphics that are output on a [monitor](https://techterms.com/definition/monitor). Desktop computers often have a [video card](https://techterms.com/definition/videocard) that contains the GPU, while mobile devices usually contain a graphics chip that is integrated into the [motherboard](https://techterms.com/definition/motherboard). By using separate processors for system and graphics processing, computers are able to handle graphic-intensive applications more efficiently.

ASSUMPTIONS

The topology of the distributed system is modeled as an undirected graph GT ¼ ðP;LÞ where P represents a set of m processors, that are assumed to be fully linked by a set of links L. An edge li;j 2 L is a bidirectional communication link between the incident processors pi and pj. The computation capacity refers to the number of operations per unit time, namely, the weight wðpiÞ of processor pi 2 P. The communication capacity refers to the data transfer rate, namely, the weight wðli;jÞ of edge li;j. If two tasks are scheduled on a single processor, then the communication cost between the two tasks is zero. For convenience of specification, let li;i be the internal communication link in the processor pi and the weight wðli;iÞ ¼ 1. The application is represented as a set of generic tasks, each linked to others by precedence constraints. The precedence constraints are first determined by analyzing the data flow among the tasks. Let G ¼ ðV; E;wÞ be a directed

acyclic graph, where V ¼ fv1; v2; . . . ; vng represents the set of tasks and E \_ V \_ V is the set of precedence constraints

among the tasks. In a situation in which an arc ei;j ¼ ði; jÞ 2 E, task vi must be completely processed before a task vj can be executed. Task vi is called a predecessor of vj, and task vj is a successor of vi. The weight wðviÞ of task vi is its computational cost and the weight wðei;jÞ of edge ei;j is its communication cost. Let Si be the set of processors that are allocated to task vi. The processing time of vi on processor px is denoted as tpðvi; pxÞ ¼ wðviÞ=wðpxÞ. The communication time of ei;j on link lk;x is denoted as tcðei;j; lk;xÞ ¼ wðei;jÞ=wðlk;xÞ. The communication time of an edge depends on the source processor and the destination processor.

Definition 1. The communication time of ei;j on link lk;x is given by tcðei;j; lk;xÞ ¼ 0 if k ¼ x, otherwise tcðei;j; lk;xÞ ¼

wðei;jÞ=wðlk;xÞ . The set of direct predecessors of task vi is represented as predðviÞ and the set of direct successors of vi is represented as succðviÞ. A task vi without any predecessor is called an entry task and a task without any successor is called an exit task. Without loss of generality, the DAG is assumed to have only one entry task ventry and only one exit task vexit. If multiple exit tasks or entry tasks exist, they may be connected with zero time-weight edges to a single pseudo-exit

task or a single pseudo-entry task that has zero time-weight.