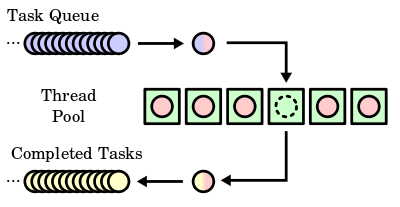
Thread pool

From Wikipedia, the free encyclopedia

[Jump to navigation](https://en.wikipedia.org/wiki/Thread_pool#mw-head)[Jump to search](https://en.wikipedia.org/wiki/Thread_pool#searchInput)

[](https://en.wikipedia.org/wiki/File:Thread_pool.svg)

A sample thread pool (green boxes) with waiting tasks (blue) and completed tasks (yellow)

In [computer programming](https://en.wikipedia.org/wiki/Computer_programming), a **thread pool** is a [software design pattern](https://en.wikipedia.org/wiki/Software_design_pattern) for achieving [concurrency](https://en.wikipedia.org/wiki/Concurrency_(computer_science)) of execution in a computer program. Often also called a **replicated workers** or **worker-crew model**,[[1]](https://en.wikipedia.org/wiki/Thread_pool#cite_note-1) a thread pool maintains multiple [threads](https://en.wikipedia.org/wiki/Thread_(computer_science)) waiting for [tasks](https://en.wikipedia.org/wiki/Task_(computers)) to be allocated for [concurrent](https://en.wikipedia.org/wiki/Concurrent_computing) execution by the supervising program. By maintaining a pool of threads, the model increases performance and avoids latency in execution due to frequent creation and destruction of threads for short-lived tasks.[[2]](https://en.wikipedia.org/wiki/Thread_pool#cite_note-2) The number of available threads is tuned to the computing resources available to the program, such as a parallel task queue after completion of execution.



**Contents**

* [1Performance](https://en.wikipedia.org/wiki/Thread_pool#Performance)
* [2See also](https://en.wikipedia.org/wiki/Thread_pool#See_also)
* [3References](https://en.wikipedia.org/wiki/Thread_pool#References)
* [4External links](https://en.wikipedia.org/wiki/Thread_pool#External_links)

Performance[[edit](https://en.wikipedia.org/w/index.php?title=Thread_pool&action=edit&section=1)]

The size of a thread pool is the number of threads kept in reserve for executing tasks. It is usually a tunable parameter of the application, adjusted to optimize program performance.[[3]](https://en.wikipedia.org/wiki/Thread_pool#cite_note-ACM,_Thread_pool_size-3) Deciding the optimal thread pool size is crucial to optimize performance. Hyperbola-based Thread-pool Analysis (HTA) technique has been suggested in order to determine optimal thread pool size for cloud-based indexing process based upon available workload and bandwidth.[[4]](https://en.wikipedia.org/wiki/Thread_pool#cite_note-IEEE,_Hyperbola-based_Thread-pool_Analysis-4)

One benefit of a thread pool over creating a new thread for each task is that thread creation and destruction overhead is restricted to the initial creation of the pool, which may result in better [performance](https://en.wikipedia.org/wiki/Performance_tuning) and better system [stability](https://en.wikipedia.org/wiki/Stability_Model). Creating and destroying a thread and its associated resources can be an expensive process in terms of time. An excessive number of threads in reserve, however, wastes memory, and context-switching between the runnable threads invokes performance penalties. A socket connection to another network host, which might take many CPU cycles to drop and re-establish, can be maintained more efficiently by associating it with a thread that lives over the course of more than one network transaction.

Using a thread pool may be useful even putting aside thread startup time. There are implementations of thread pools that make it trivial to queue up work, control concurrency and sync threads at a higher level than can be done easily when manually managing threads[[5]](https://en.wikipedia.org/wiki/Thread_pool#cite_note-5)[[6]](https://en.wikipedia.org/wiki/Thread_pool#cite_note-6). In these cases the performance benefits of use may be secondary.

Typically, a thread pool executes on a single computer. However, thread pools are conceptually related to [server farms](https://en.wikipedia.org/wiki/Server_farm) in which a master process, which might be a thread pool itself, distributes tasks to worker processes on different computers, in order to increase the overall throughput. [Embarrassingly parallel](https://en.wikipedia.org/wiki/Embarrassingly_parallel) problems are highly amenable to this approach.[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

The number of threads may be dynamically adjusted during the lifetime of an application based on the number of waiting tasks. For example, a [web server](https://en.wikipedia.org/wiki/Web_server) can add threads if numerous [web page](https://en.wikipedia.org/wiki/Web_page) requests come in and can remove threads when those requests taper down.[[*disputed*](https://en.wikipedia.org/wiki/Wikipedia:Disputed_statement)*–*[*discuss*](https://en.wikipedia.org/wiki/Talk:Thread_pool)] The cost of having a larger thread pool is increased resource usage. The algorithm used to determine when to create or destroy threads affects the overall performance:

* Creating too many threads wastes resources and costs time creating the unused threads.
* Destroying too many threads requires more time later when creating them again.
* Creating threads too slowly might result in poor client performance (long wait times).
* Destroying threads too slowly may starve other processes of resources.

See also[[edit](https://en.wikipedia.org/w/index.php?title=Thread_pool&action=edit&section=2)]

* [Asynchrony (computer programming)](https://en.wikipedia.org/wiki/Asynchrony_(computer_programming))
* [Object pool pattern](https://en.wikipedia.org/wiki/Object_pool_pattern)
* [Concurrency pattern](https://en.wikipedia.org/wiki/Concurrency_pattern)
* [Grand Central Dispatch](https://en.wikipedia.org/wiki/Grand_Central_Dispatch)
* [Parallel Extensions](https://en.wikipedia.org/wiki/Parallel_Extensions)
* [Parallelization](https://en.wikipedia.org/wiki/Parallelization)
* [Server farm](https://en.wikipedia.org/wiki/Server_farm)
* [Staged event-driven architecture](https://en.wikipedia.org/wiki/Staged_event-driven_architecture)

References[[edit](https://en.wikipedia.org/w/index.php?title=Thread_pool&action=edit&section=3)]

* 1. [**^**](https://en.wikipedia.org/wiki/Thread_pool#cite_ref-1) Garg, Rajat P. & Sharapov, Ilya *Techniques for Optimizing Applications - High Performance Computing* Prentice-Hall 2002, p. 394
  2. [**^**](https://en.wikipedia.org/wiki/Thread_pool#cite_ref-2) [*Holub, Allen*](https://en.wikipedia.org/wiki/Allen_Holub)*(2000). Taming Java Threads. Apress. p. 209.*
  3. [**^**](https://en.wikipedia.org/wiki/Thread_pool#cite_ref-ACM,_Thread_pool_size_3-0) *Yibei Ling; Tracy Mullen; Xiaola Lin (April 2000). "Analysis of optimal thread pool size". ACM SIGOPS Operating Systems Review.****34****(2): 42–55.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1145/346152.346320*](https://doi.org/10.1145%2F346152.346320)*.*
  4. [**^**](https://en.wikipedia.org/wiki/Thread_pool#cite_ref-IEEE,_Hyperbola-based_Thread-pool_Analysis_4-0) *Harshit Gujral; Abhinav Sharma; Sangeeta Mittal (August 2018). "Determination of Optimal Thread Pool for Cloud Based Concurrent Enhanced No-Escape Search". 2018 Eleventh International Conference on Contemporary Computing (IC3). pp. 1–6.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1109/IC3.2018.8530645*](https://doi.org/10.1109%2FIC3.2018.8530645)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-1-5386-6834-4*](https://en.wikipedia.org/wiki/Special:BookSources/978-1-5386-6834-4)*.*
  5. [**^**](https://en.wikipedia.org/wiki/Thread_pool#cite_ref-5) [*"QThreadPool Class | Qt Core 5.13.1"*](https://doc.qt.io/qt-5/qthreadpool.html)*.*
  6. [**^**](https://en.wikipedia.org/wiki/Thread_pool#cite_ref-6) [*"GitHub - vit-vit/CTPL: Modern and efficient C++ Thread Pool Library"*](https://github.com/vit-vit/ctpl)*. 2019-09-24.*

External links[[edit](https://en.wikipedia.org/w/index.php?title=Thread_pool&action=edit&section=4)]

* "[Query by Slice, Parallel Execute, and Join: A Thread Pool Pattern in Java](https://web.archive.org/web/20080207124322/http:/today.java.net/pub/a/today/2008/01/31/query-by-slice-parallel-execute-join-thread-pool-pattern.html)" by Binildas C. A.
* "[Thread pools and work queues](http://www.ibm.com/developerworks/java/library/j-jtp0730.html)" by Brian Goetz
* "[A Method of Worker Thread Pooling](https://www.codeproject.com/Articles/3631/A-Method-of-Worker-Thread-Pooling)" by Pradeep Kumar Sahu
* "[Work Queue](https://www.codeproject.com/Articles/3607/Work-Queue)" by Uri Twig: C++ code demonstration of pooled threads executing a work queue.
* "[Windows Thread Pooling and Execution Chaining](http://www.codeproject.com/Articles/6863/Windows-Thread-Pooling-and-Execution-Chaining)"
* "[Smart Thread Pool](http://www.codeproject.com/KB/threads/smartthreadpool.aspx)" by Ami Bar
* "[Programming the Thread Pool in the .NET Framework](http://msdn.microsoft.com/en-us/library/ms973903.aspx)" by David Carmona
* "[The Thread Pool and Asynchronous Methods](http://www.yoda.arachsys.com/csharp/threads/threadpool.shtml)" by Jon Skeet
* "[Creating a Notifying Blocking Thread Pool in Java](http://today.java.net/pub/a/today/2008/10/23/creating-a-notifying-blocking-thread-pool-executor.html)" by Amir Kirsh
* "[Practical Threaded Programming with Python: Thread Pools and Queues](http://www.ibm.com/developerworks/aix/library/au-threadingpython/)" by Noah Gift
* "[Optimizing Thread-Pool Strategies for Real-Time CORBA](http://www.cs.wustl.edu/~schmidt/PDF/OM-01.pdf)" by Irfan Pyarali, Marina Spivak, [Douglas C. Schmidt](https://en.wikipedia.org/wiki/Douglas_C._Schmidt) and Ron Cytron
* Conference Paper "[Deferred cancellation. A behavioral pattern](http://doi.acm.org/10.1145/1753196.1753218)" by Philipp Bachmann