Grand Central Dispatch

From Wikipedia, the free encyclopedia

Grand Central Dispatch (**GCD**) is a technology developed by Apple Inc. to optimize application support for systems with multi-core processors and other symmetric multiprocessing systems.[2] It is an implementation of task parallelism based on the thread pool pattern. The fundamental idea is to move the management of the thread pool out of the hands of the developer, and closer to the operating system. The developer injects "work packages" into the pool oblivious of the pool's architecture. This model improves simplicity, portability and performance.

GCD was first released with Mac OS X 10.6, and is also available with iOS 4 and above. The name "Grand Central Dispatch" is a reference to Grand Central Terminal.

Grand Central Dispatch

Developer(s)	Apple Inc.
Operating system	Mac OS X 10.6 and later, iOS 4.0 and later,[1] FreeBSD
Type	System Utility
License	Apache 2.0
Website	https://libdispatch.macosforge.org

The source code for the library that provides the implementation of GCD's services, **libdispatch**, was released by Apple under the Apache License on September 10, 2009[3] It has been ported[4] to the FreeBSD operating system, starting with FreeBSD 8.1.[5] MidnightBSD 0.3-CURRENT includes "libdispatch" without blocks support.[6] Linux and Solaris support are provided within the upstream trunk.[7][8] In order to develop support for Windows, currently two forks exist at opensource.mlba-team.de (http://opensource.mlba-team.de/xdispatch) and Github (https://github.com/DrPizza/libdispatch).[9][10] Apple has its own port of libdispatch.dll for Windows shipped with Safari and iTunes, but no SDK is provided.

Contents

Design

GCD works by allowing specific tasks in a program that can be run in parallel to be queued up for execution and, depending on availability of processing resources, scheduling them to execute on any of the available processor cores[11][12] (referred to as "routing" by Apple).[13]

A task can be expressed either as a function or as a "block."[14] Blocks are an extension to the syntax of C, C++, and Objective-C programming languages that encapsulate code and data into a single object in a way similar to a closure.[11] GCD can still be used in environments where blocks are not available.

Grand Central Dispatch still uses threads at the low level but abstracts them away from the programmer, who will not need to be concerned with as many details. Tasks in GCD are lightweight to create and queue; Apple states that 15 instructions are required to queue up a work unit in GCD, while creating a traditional thread could easily require several hundred instructions.[11]

A task in Grand Central Dispatch can be used either to create a work item that is placed in a queue or assign it to an event source. If a task is assigned to an event source, then a work unit is made from the block or function when the event triggers, and the work unit is placed in an appropriate queue. This is described by Apple as more efficient than creating a thread whose sole purpose is to wait on a single event triggering.

Features

The dispatch framework declares several data types and functions to create and manipulate them:

- *Dispatch Queues* are objects that maintain a queue of *tasks*, either anonymous code blocks or functions, and execute these tasks in their turn. The library automatically creates several queues with different priority levels that execute several tasks concurrently, selecting the optimal number of tasks to run based on the operating environment. A client to the library may also create any number of serial queues, which execute tasks in the order they are submitted, one at a time.[12] Because a serial queue can only run one task at a time, each task submitted to the queue is critical with regard to the other tasks on the queue, and thus a serial queue can be used instead of a lock on a contended resource.
- *Dispatch Sources* are objects that allow the client to register blocks or functions to execute asynchronously upon system events, such as a socket or file descriptor being ready for reading or writing, or a POSIX signal.
- *Dispatch Groups* are objects that allow several tasks to be grouped for later joining. Tasks can be added to a queue as a member of a group, and then the client can use the group object to wait until all of the tasks in that group have completed.
- *Dispatch Semaphores* are objects that allow a client to permit only a certain number of tasks to execute concurrently.

Examples

Two examples that demonstrate the use of Grand Central Dispatch can be found in John Siracusa's *Ars Technica* Snow Leopard review.[15] Initially, a document-based application has a method called `analyzeDocument` which may do something like count the number of words and paragraphs in the document. Normally, this would be a quick process, and may be executed in the main thread without the user noticing a delay between pressing a button and the results showing.

```
- (IBAction)analyzeDocument:(NSButton *)sender {
    NSDictionary *stats = [myDoc analyze];
    [myModel setDict:stats];
    [myStatsView setNeedsDisplay:YES];
  }
```

If the document is large and analysis takes a long time to execute then the main thread will wait for the function to finish. If it takes long enough, the user will notice, and the application may even "beachball". The solution can be seen here:

```
- (IBAction)analyzeDocument:(NSButton *)sender {
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
        NSDictionary *stats = [myDoc analyze];
        dispatch_async(dispatch_get_main_queue(), ^{
          [myModel setDict:stats];
          [myStatsView setNeedsDisplay:YES];
        });
    });
}
```

Here, the call to `[myDoc analyze]` is placed inside a Block, which is then placed on one of the global concurrent queues. After it has finished running `[myDoc analyze]`, a new block is placed on the main queue (on which the main thread of the application runs), which updates the GUI (This is necessary because the GUI can only be updated by the main thread). By making these two small changes, the developer has avoided a potential stall of the application as seen by the user, and allowed their application to make better use of hardware resources.

The second example is that of parallelising a for loop:

```
for (i = 0; i < count; i++) {
      results[i] = do_work(data, i);
}
total = summarize(results, count);
```

This code runs the `do_work` function `count` times, assigning the i_{th} result to the i_{th} element in the array `results`, and then calls summarize on array once the loop has ended. Unfortunately the work is computed sequentially, where it may not need to be. Assuming that do_work doesn't rely on the results of any of the other calls made to it, there is no reason why these calls cannot be made concurrently. This is how this would be done in GCD:

```
dispatch_apply(count, dispatch_get_global_queue(0, 0), ^(size_t i){
```

```
    results[i] = do_work(data, i);
  });
total = summarize(results, count);
```

Here, `dispatch_apply` runs the block passed to it, `count` times, placing each invocation on a global queue, and passing each block invocation a different number from 0 to `count`-1. This will allow the OS to spread out the work as it sees fit, choosing the optimal number of threads to run on for the current hardware and system load. `dispatch_apply` does not return until all the blocks it places on the given queue have completed execution, so that it can be guaranteed that all the work inside the original loop has completed before calling `summarize`.

Programmers can create their own serial queues for tasks which they know must run serially but which may be executed on a separate thread. A new queue would be created like so:

```
dispatch_queue_t exampleQueue;
exampleQueue = dispatch_queue_create( "com.example.unique.identifier", NULL );

// exampleQueue may be used here.

dispatch_release( exampleQueue );
```

Care must be taken to avoid a dispatched block on a queue synchronously placing another block on the same queue as this is guaranteed to deadlock. Such code might do the following:

```
dispatch_queue_t exampleQueue = dispatch_queue_create( "com.example.unique.identifier", NULL );

dispatch_sync( exampleQueue, ^{
  dispatch_sync( exampleQueue, ^{
    printf( "I am now deadlocked...\n" );
  });
});

dispatch_release( exampleQueue );
```

Applications

GCD is used throughout OS X (beginning with 10.6 Snow Leopard), and Apple has encouraged its adoption by OS X application developers. FreeBSD developer Robert Watson announced the first adaptation of a major open source application, the Apache HTTP Server, to use GCD via the Apache GCD MPM (Multi-Processing Module) on May 11, 2010, in order to illustrate the programming model and how to integrate GCD into existing, large-scale multi-threaded, applications. His announcement observed that the GCD MPM had ⅓–½ the number of lines as other threaded MPMs.[16][17]

Internals

GCD is implemented by libdispatch, with support from pthreads non-POSIX extensions developed by Apple. Apple has changed the interface since its inception (in OS X 10.5) through the official launch of GCD (10.6), Mountain Lion (10.8) and recently Mavericks (10.9). The latest changes involve making

the code supporting pthreads, both in user mode and kernel, private (with kernel pthread support reduced to shims only, and the actual implementation moved to a separate kernel extension).[18]

See also

- Task Parallel Library
- Java Concurrency
- OpenMP
- Threading Building Blocks (TBB)

References

1. ^ "Grand Central Dispatch (GCD) Reference" (http://developer.apple.com/mac/library/documentation/Performance/Reference/GCD_libdispatch_Ref/Reference/reference.html). Apple Inc.
2. ^ Apple Previews Mac OS X Snow Leopard to Developers (http://www.apple.com/pr/library/2008/06/09snowleopard.html), June 9, 2008.
3. ^ http://libdispatch.macosforge.org/
4. ^ GCD libdispatch w/Blocks support working on FreeBSD (http://lists.macosforge.org/pipermail/libdispatch-dev/2009-September/000059.html)
5. ^ FreeBSD Quarterly Status Report (http://www.freebsd.org/news/status/report-2009-04-2009-09.html#Grand-Central-Dispatch---FreeBSD-port)
6. ^ libdispatch (http://www.justjournal.com/users/mbsd/entry/18349)
7. ^ libdispatch mailing list: "Porting status Linux" (http://lists.macosforge.org/pipermail/libdispatch-dev/2011-April/000485.html) April 10, 2011
8. ^ libdispatch mailing list: "Porting status Solaris x86/64" (http://lists.macosforge.org/pipermail/libdispatch-dev/2011-April/000486.html) April 10, 2011
9. ^ libdispatch mailing list: "libdispatch for Win32" (http://lists.macosforge.org/pipermail/libdispatch-dev/2011-April/000510.html) April 22, 2011
10. ^ libdispatch mailing list: "Updates regarding the status of libdispatch on Windows" (http://lists.macosforge.org/pipermail/libdispatch-dev/2011-May/000515.html) May 5, 2011
11. ^ ***a*** ***b*** ***c*** Apple Technical Brief on Grand Central Dispatch (https://web.archive.org/web/20090920043909/http://images.apple.com/macosx/technology/docs/GrandCentral_TB_brief_20090903.pdf) at the Wayback Machine (archived September 20, 2009)
12. ^ ***a*** ***b*** Gagne, Abraham Silberschatz, Peter Baer Galvin, Greg (2013). *Operating system concepts* (9th ed. ed.). Hoboken, N.J.: Wiley. p. 182-183. ISBN 9781118063330.
13. ^ "WWDC 2008: New in Mac OS X Snow Leopard" (http://www.roughlydrafted.com/2008/06/12/wwdc-2008-new-in-mac-os-x-snow-leopard/). Retrieved June 18, 2008.
14. ^ "Grand Central Dispatch (GCD) Reference" (http://developer.apple.com/mac/library/documentation/Performance/Reference/GCD_libdispatch_Ref/Reference/reference.html). Retrieved September 13, 2009.
15. ^ Mac OS X 10.6 Snow Leopard: the Ars Technica review (http://arstechnica.com/apple/reviews/2009/08/mac-os-x-10-6.ars/13) (accessed September 2, 2009)

16. ^ libdispatch-dev GCD MPM for Apache (http://lists.macosforge.org/pipermail/libdispatch-dev/2010-May/000352.html) (accessed May 14, 2010)
17. ^ apache-libdispatch (http://libdispatch.macosforge.org/trac/wiki/apache) (accessed May 14, 2010)
18. ^ Levin, Jonathan (15 February 2014). "GCD Internals: The undocumented side of the Grand Central Dispatcher" (http://newosxbook.com/articles/GCD.html). Retrieved 17 March 2014.

External links

- GCD project on Mac OS Forge (https://libdispatch.macosforge.org)
- GCD Reference from the Mac Developer Library (https://developer.apple.com/library/mac/#documentation/Performance/Reference/GCD_libdispatch_Ref/Reference/reference.html)
- The *Introducing Blocks and Grand Central Dispatch* article from the Mac Developer Library (https://developer.apple.com/library/mac/#featuredarticles/BlocksGCD/_index.html)