Thread Pools

A thread pool is a collection of threads that the Ice run time draws upon to perform specific tasks.

On this page:

Introduction to Thread Pools

Each communicator creates two thread pools:

- The *client thread pool* services outgoing connections, which primarily involves handling the replies to outgoing requests and includes notifying AMI callback objects. If a connection is used in bidirectional mode, the client thread pool also dispatches incoming callback requests.
- The *server thread pool* services incoming connections. It dispatches incoming requests and, for bidirectional connections, processes replies to outgoing requests.

By default, these two thread pools are shared by all of the communicator's object adapters. If necessary, you can configure individual object adapters to use a private thread pool instead.

If a thread pool is exhausted because all threads are currently dispatching a request, additional incoming requests are transparently delayed until a request completes and relinquishes its thread; that thread is then used to dispatch the next pending request. Ice minimizes thread context switches in a thread pool by using a leader-follower implementation [1].

Configuring Thread Pools

Each thread pool has a unique name that serves as the prefix for its configuration properties:

- *`name`*`.Size`
  This property specifies the initial size of the thread pool. If not defined, the default value is one.

- *`name`*`.SizeMax`
  This property specifies the maximum size of the thread pool. If not defined, the default value is one. If the value of this property is less than that of *`name`*`.Size`, this property is adjusted to be equal to *`name`*`.Size`.

- *`name`*`.SizeWarn`
  This property sets a high water mark; when the number of threads in a pool reaches this value, the Ice run time logs a warning message. If you see this warning message frequently, it could indicate that you need to increase the value of *`name`*`.SizeMax`. The default value is zero, which disables the warning.

- *`name`*`.StackSize`
  This property specifies the number of bytes to use as the stack size of threads in the thread pool. The operating system's default is used if this property is not defined or is set to zero.

- *`name`*`.Serialize`
  Setting this property to a value greater than zero forces the thread pool to serialize all messages received over a connection. It is unnecessary to enable serialization for a thread pool whose maximum size is one because such a thread pool is already limited to processing one message at a time. For thread pools with more than one thread, serialization has a negative impact on latency and throughput. If not defined, the default value is zero. We discuss this feature in more detail in Thread Pool Design Considerations.

- *`name`*`.ThreadIdleTime`
  This property specifies the number of seconds that a thread in the thread pool must be idle before it terminates. The default value is 60 seconds if this property is not defined. Setting it to zero disables the termination of idle threads.

For configuration purposes, the names of the client and server thread pools are `Ice.ThreadPool.Client` and `Ice.ThreadPool.Server`, respectively. As an example, the following properties establish the initial and maximum sizes for these thread pools:

```
Ice.ThreadPool.Client.Size=1
Ice.ThreadPool.Client.SizeMax=10
Ice.ThreadPool.Server.Size=1
Ice.ThreadPool.Server.SizeMax=10
```

To monitor the thread pool activities of the Ice run time, you can enable the `Ice.Trace.ThreadPool` property. Setting this property to a non-zero value causes the Ice run time to log a message when it creates a thread pool, as well as each time the size of a thread pool increases or decreases.

Dynamic Thread Pools

A *dynamic* thread pool can grow and shrink when necessary in response to changes in an application's work load. All thread pools have at least one thread, but a dynamic thread pool can grow as the demand for threads increases, up to the pool's maximum size. Threads may also be terminated automatically when they have been idle for some time.

The dynamic nature of a thread pool is determined by the configuration properties `name.Size`, `name.SizeMax`, and `name.ThreadIdleTime`. A thread pool is not dynamic in its default configuration because `name.Size` and `name.SizeMax` are both set to one, meaning the pool can never grow to contain more than a single thread. To configure a dynamic thread pool, you must set at least one of `name.Size` or `name.SizeMax` to a value greater than one. We can use several configuration scenarios to explore the semantics of dynamic thread pools in greater detail:

- `name.SizeMax=5`

  This thread pool initially contains a single thread because `name.Size` has a default value of one, and Ice can grow the pool up to the maximum of five threads. During periods of inactivity, idle threads terminate after 60 seconds (the default value for `name.ThreadIdleTime`) until the pool contains just one thread again.

- `name.Size=3`
  `name.SizeMax=5`

  This thread pool starts with three active threads but otherwise behaves the same as in the previous configuration. The pool can still shrink to a size of one as threads become idle.

- `name.Size=3`
  `name.ThreadIdleTime=10`

  This thread pool starts with three active threads and shrinks quickly to one thread during periods of inactivity. As demand increases again, the thread pool can return to its maximum size of three threads (`name.SizeMax` defaults to the value of `name.Size`).

- `name.SizeMax=5`
  `name.ThreadIdleTime=0`

  This thread pool can grow from its initial size of one thread to contain up to five threads, but it will never shrink because `name.ThreadIdleTime` is set to zero.

- `name.Size=5`
  `name.ThreadIdleTime=0`

  This thread pool starts with five threads and can neither grow nor shrink.

To summarize, the value of `name.ThreadIdleTime` determines whether (and how quickly) a thread pool can shrink to a size of one. A thread pool that shrinks can also grow to its maximum size. Finally, setting `name.SizeMax` to a value larger than `name.Size` allows a thread pool to grow beyond its initial capacity.

See Also

- Thread Pool Design Considerations
- Bidirectional Connections
- Object Adapters
- Object Adapter Thread Pools
- Ice Thread Pool Properties

References

1. Schmidt, D. C. et al. 2000. "Leader/Followers: A Design Pattern for Efficient Multi-Threaded Event Demultiplexing and Dispatching". In *Proceedings of the 7th Pattern Languages of Programs Conference*, WUCS-00-29, Seattle, WA: University of Washington.