

CW2001

ACM SIGPLAN Continuation Workshop 2011

Swarm

Transparent Scalability Through Portable Continuations

<https://github.com/sanity/Swarm>

Submitted on:

Jul 2, 2011

Submitted by:

James Douglas

james@earldouglas.com

Abstract

Transparent scalability is an elusive characteristic sought for successful software projects which inevitably outgrow themselves. A common way to approach the design of such applications is with the MapReduce pattern, which requires considerable foresight into how the application can be broken down into the functional map and reduce operations. A problem with this and similar approaches is the investment required at the beginning of development; the problem domain must be carefully analyzed and a solution crafted to support the predicted scalability needs. It would be preferable if applications could be developed simply and cheaply, then later, when necessary, made scalable without reworking the existing source code. We present an approach to building transparently scalable applications using Swarm, a framework which enables code execution to "follow the data" within Scala's serializable delimited continuations. Swarm abstracts the location of data across a distributed system from the developer, eliminating costly architectural and modeling requirements of popular distributed computing patterns and frameworks. We explain the design of an example implementation of a Twitter-like Web application which uses Swarm's continuation-passing style collections, and show how the developer is unburdened by the complexity of scalability. We demonstrate how this Swarm-based application can be transparently scaled without requiring changes to the code or accommodation by the architecture.

Supplementary material

Code selections

The following code represents a selection from the Swarm source code hosted on GitHub. It includes a loop which listens for incoming serialized continuations from remote nodes, and sends them off to be dereferenced, executed, and/or relocated as appropriate.

Swarm's listen loop:

```
val server = new java.net.ServerSocket(port);

var runnable = new Runnable() {
  override def run() = {
    while (true) {
      val socket = server.accept()
      val ois = new java.io.ObjectInputStream(socket.getInputStream())
      val bee = ois.readObject().asInstanceOf[(Unit => Bee)]
      debug("resuming execution from " + local)
      Swarm.continue(bee)
    }
  }
}
Swarm.executor.execute(runnable)
```

Swarm's continue method:

```
def continue(f: Unit => Bee)(implicit tx: Transporter) {
  execute(reset(f()))
}
```

Swarm's execute method:

```
def execute(bee: Bee)(implicit tx: Transporter) {
  bee match {
    case RefBee(f, ref) if (tx.isLocal(ref.location)) =>
      if (!Store.exists(ref.uid)) {
        val newRef = Store.relocated(ref.uid)
        ref.relocate(newRef.uid, newRef.location)
        tx.transport(f, ref.location)
      } else {
        Swarm.continue(f)
      }
    case RefBee(f, ref) => tx.transport(f, ref.location)
    case IsBee(f, destination) if (tx.isLocal(destination)) =>
      Swarm.continue(f)
    case IsBee(f, destination) => tx.transport(f, destination)
    case NoBee() =>
  }
}
```

Demo screenshot

The following screenshot shows the main view of the Swarm Twitter demo after statuses have been submitted by different users. These statuses are made available on multiple server nodes via Swarm.

Swarm Twitter main view:

