Actor Bug: Concurrency - Shutdown

Every system must shut down at some point and should do so gracefully. However, that is not always the case and this category represent the bugs involving a problematic shutdown process.

For more complex systems a graceful shut down is essential. There was a bug in Collossus where the system would start to terminate and it would not only close some of it's processes down. It would close the delegators but leave existing connections open to linger for a time. The issue was solved by having the Worker actors check if it's server connection and delegator connection were both closed and then unregistered the connection itself as required.

This issue was fixed by giving the worker a more robust approach to closing connections. They do this by running a collect function on the connections of a worker. If the connection is equal to the delagator server than the connection is closed with an unregisterConnection call.

Link: https://github.com/tumblr/colossus/issues/111

Code:

colossus/core/Worker.scala

```
def unregisterServer(handler: ActorRef) {
     if (delegators contains handler) {
       val delegator = delegators(handler)
-      log.info(s"unregistering server ${delegator.server.name}")
+      val closed = connections.collect{
+        case (_, s: ServerConnection) if (s.server == delegator.server) => {
+          unregisterConnection(s, DisconnectCause.Terminated)
+        }
+      }
       delegators -= handler
       delegator.onShutdown()
+      log.info(s"unregistering server ${delegator.server.name} (terminating ${closed.size} associated
connections)")
     } else {
       log.warning(s"Attempted to unregister unknown server actor ${handler.path.toString}")
```