Principles of Software Construction

Introduction to distributed systems

Josh Bloch  Charlie Garrod
Administrivia

- Homework 5 Best Frameworks available tonight
Key concepts from Tuesday
Concurrence at the language level

• Consider:
  
```java
Collection<Integer> collection = ...;
int sum = 0;
for (int i : collection) {
    sum += i;
}
```

• In python:
  
```python
collection = ...
sum = 0
for item in collection:
    sum += item
```
Parallel prefix sums algorithm, upsweep

- Computes the partial sums in a more useful manner

\[
\begin{array}{cccccccc}
13 & 9 & -4 & 19 & -6 & 2 & 6 & 3 \\
13 & 22 & -4 & 15 & -6 & -4 & 6 & 9 \\
13 & 22 & -4 & 37 & -6 & -4 & 6 & 5 \\
13 & 22 & -4 & 37 & -6 & -4 & 6 & 42 \\
\vdots
\end{array}
\]
Parallel prefix sums algorithm, downsweep

- Now unwinds to calculate the other sums

\[
\begin{bmatrix}
13, & 22, & -4, & 37, & -6, & -4, & 6, & 42 \\
13, & 22, & -4, & 37, & -6, & 33, & 6, & 42 \\
13, & 22, & 18, & 37, & 31, & 33, & 39, & 42 \\
\end{bmatrix}
\]

- Recall, we started with:

\[
\begin{bmatrix}
13, & 9, & -4, & 19, & -6, & 2, & 6, & 3 \\
\end{bmatrix}
\]
A framework for asynchronous computation

• The `java.util.concurrent.Future<V>` interface:
  
  V get();
  V get(long timeout, TimeUnit unit);
  boolean isDone();
  boolean cancel(boolean mayInterruptIfRunning);
  boolean isCancelled();

• The `java.util.concurrent.ExecutorService` interface:
  
  Future<?> submit(Runnable task);
  Future<V> submit(Callable<V> task);
  List<Future<V>> invokeAll(Collection<? extends Callable<V>> tasks);
  Future<V> invokeAny(Collection<? extends Callable<V>> tasks);
  void shutdown();
Fork/Join: another common computational pattern

• In a long computation:
  – Fork a thread (or more) to do some work
  – Join the thread(s) to obtain the result of the work

• The `java.util.concurrent.ForkJoinPool` class
  – Implements `ExecutorService`
  – Executes `java.util.concurrent.ForkJoinTask<V>` or
    `java.util.concurrent.RecursiveTask<V>` or
    `java.util.concurrent.RecursiveAction`
Parallel prefix sums algorithm

- How good is this?
  - Work: O(n)
  - Depth: O(lg n)
- See PrefixSumsParallelArrays.java
- See PrefixSumsSequential.java
  - n-1 additions
  - Memory access is sequential
- For PrefixSumsSequentialWithParallelWork.java
  - About 2n useful additions, plus extra additions for the loop indexes
  - Memory access is non-sequential
- The punchline:
  - Don't roll your own
  - Cache and constants matter
Today: Distributed system design

• Java networking fundamentals
• Introduction to distributed systems
  – Challenges
  – Motivation: reliability and scalability
  – Failure models
Our destination: Distributed systems

- Multiple system components (computers) communicating via some medium (the network)
- Challenges:
  - Scale
  - Concurrency
  - Heterogeneity
  - Geography
  - Failures
  - Security

(courtesy of http://www.cs.cmu.edu/~dga/15-440/F12/lectures/02-internet1.pdf)
Abstraction: Communication protocols

- Agreement between parties for how communication should take place

(Friendly greeting.
Muttered reply.
Destination?
Pittsburgh.
Thank you.)

(courtesy of http://www.cs.cmu.edu/~dga/15-440/F12/lectures/02-internet1.pdf)
Abstractions of a network connection
Internet addresses and sockets

- For IP version 4 (IPv4) host address is a 4-byte number
  - e.g. 127.0.0.1
  - Hostnames mapped to host IP addresses via DNS
  - ~4 billion distinct addresses

- Port is a 16-bit number (0-65535), assigned conventionally
  - e.g., port 80 is the standard port for web servers
Packet-oriented and stream-oriented connections

- **UDP**: User Datagram Protocol
  - Unreliable, discrete packets of data
- **TCP**: Transmission Control Protocol
  - Reliable data stream
TCP, networking in Java

- The java.net.InetAddress:
  
  static InetAddress getByName(String host);
  static InetAddress getByAddress(byte[] b);
  static InetAddress getLocalHost();

- The java.net.Socket:
  
  Socket(InetAddress addr, int port);
  boolean isConnected();
  boolean isClosed();
  void close();
  InputStream getInputStream();
  OutputStream getOutputStream();

- The java.net.ServerSocket:
  
  ServerSocket(int port);
  Socket accept();
  void close();

  ...
Simple sockets demo

• NetworkServer.java
• A basic chat system:
  – TransferThread.java
  – TextSocketClient.java
  – TextSocketServer.java
Higher levels of abstraction

- Application-level communication protocols
- Frameworks for remote computation
  - Remote Procedure Call (RPC)
  - Java Remote Method Invocation (RMI)
- Common distributed system architectures and primitives
  - e.g., distributed consensus, transactions, replication
- Complex computational frameworks
  - e.g., distributed map-reduce
Today: Distributed system design

• Java networking fundamentals
• Introduction to distributed systems
  – Challenges
  – Motivation: reliability and scalability
  – Failure models
Metrics of success

- **Reliability**
  - Often in terms of availability: fraction of time system is working
    - 99.999% available is "5 nines of availability"

- **Scalability**
  - Ability to handle workload growth
You need to restart your computer. Hold down the Power button for several seconds or press the Restart button.

Veuillez redémarrer votre ordinateur. Maintenez la touche de démarrage enfoncée pendant plusieurs secondes ou bien appuyez sur le bouton de réinitialisation.

Sie müssen Ihren Computer neu starten. Halten Sie dazu die Einschalttaste einige Sekunden gedrückt oder drücken Sie die Neustart-Taste.

コンピュータを再起動する必要があります。パワーボタンを数秒間押し続けるか、リセットボタンを押してください。
Types of failure behaviors

• Fail-stop
• Other halting failures
• Communication failures
  – Send/receive omissions
  – Network partitions
  – Message corruption
• Data corruption
• Performance failures
  – High packet loss rate
  – Low throughput
  – High latency
• Byzantine failures
Common assumptions about failures

- Behavior of others is fail-stop (ugh)
- Network is reliable (ugh)
- Network is semi-reliable but asynchronous
- Network messages are not corrupt
- Network failures are transitive
- Failures are independent
- Local data is not corrupt
- Failures are reliably detectable
- Failures are unreliably detectable
Summary

• Distributed systems are a great source of complexity
  – Abstractions to reduce complexity:
    • Protocols
    • RPC and computational frameworks
    • Common building blocks

• Tuesday: Distributed map reduce