1. **Explain the differences in *job* execution times.**

As you can see jobs run slightly faster when executed on the same JVM, compared to different. This is because the Computer and Space can communicate within the same JVM, instead of communicating between JVM's. When computer(s) are running on running on different physical machines execution time increases. This is due to the fact that every task has to be transported back and forth across the LAN. When two computers are used, execution time is almost halved, this is because the computer's are run on different physical machines, effectively running in parallel.

2. **To improve your understanding of what is going on, what other measurements would you like to see included?**

It would have been useful to measure execution time of the tasks execute method. This way we could isolate actual execution form network latency. Doing so would also allow us to measure the fraction of job execution time that is ue to network latency. Other measures that could be interesting is computer ideal time, and task queue size.

3. **Enumerate what you believe to be the key points concerning thread safety.**

A key point concerning thread safety, is that the fields in a object/class has a valid state. To manage this you can make objects immutable, have critical sections synchronized and use a wrapper that is thread-safe.

An object is immutable if its state can't be changed after it is initiated. It avoids any read-write or write-write conflict, due it can't be changed by a thread.

A critical section, is where only one thread can execute an operation at a time, as a single operation. To ensure this, you can use the javas keyword *synchronized.*

It is also possible to make a thread-safe wrapper class. When embedding the original class in a thread-safe wrapper class, instances of the original class can use the wrappers instances as "front-ends". This will not change the original class, and it will function as thread safe.