COMS W4115 Tandem White Paper

Team 11

2/21/12

Abdullah Al-Syed, aza2105

Jenee Benjamin, jlb2229

Patrick De La Garza, pmd2130

Donald Pomeroy, dep2127

Varun Ravishankar, vr2263

1 Introduction to Tandem

State machines are fundamental to how we program, from using regular expressions to creating simulations. However, it is often difficult to create state machines without creating spaghetti code. State machine code is riddled with 3 or more nested levels of if-else statement, calling functions that might be anywhere in the code. While libraries have attempted to solve this problem, most programmers prefer to avoid creating state machines, even when it might be a natural description for the algorithm at hand. Tandem attempts to solve these problems by taking inspiration from finite state automata and abstract state machines by making the node a central feature in the language, and letting the programmer focus on the inputs to be handled, the outputs that are produced, and transitions to the next state. This leads to a easy-to-read syntax for state machines and makes it easy to introduce concurrency to handle the large simulations that will be created on multicore machines in the near future.

Tandem is well-suited for simulations of any type, including hardware, networks, and physics simulations. Evolutionary programs can be described as a set of states with a feedback loop, converging on a final output; genetic algorithms and neural networks can be described with simple Tandem programs. Finally, algorithms that are best described with state machines, like the error recovery in TCP network programming, are awkward to program in most languages. Tandem, however, can handle the coupling between chains of states without confusing the programmer about the transitions between states. If a

program can be described as a series of transitions from one state to another, it is suited for Tandem. Programmers will find that using Tandem produces easy-to-read and maintainable code without forcing them to use abstractions that are not natural to the problem.

2 Main Features of Tandem

Tandem is based upon the concept of the node, representing a state that the program is in. Nodes can be linked together to form chains of nodes, which in turn can be led to be thought of as nodes. This leads to easy modularity and object-oriented programming. Tandem lends itself to parallel programming, with no data being shared between nodes and easy simulation of non-deterministic state machines leading to massively concurrent programs.

2.1 Nodal

The Tandem Language is nodal. This is Tandem's fundamental trait and makes possible the language's capabilities. In Tandem, everything is a Node. Tandem code can be thought of as a directed graph of functions or states (nodes). Every program written in Tandem is itself a node, even if it is composed of a larger chain of nodes. The nodal logical-structure underlying Tandem makes possible a combination of traits that allows the user to comfortably tackle a wide variety of problems and work in a paradigm that is conducive to easy and innovative development.

2.2 Object-oriented

Tandem is an object-oriented language. Each node in Tandem forms an object with its own set of variables and functions hidden from other nodes that it can call on to process its input. Nodes can also be linked into a collection of interconnected nodes, or a super node, that can also be treated as a single node object that takes a set of inputs, processes them, and produces some output. Only the output of the nodes operation and the users input is ever passed from one node to another. Tandem has a built-in sense of encapsulation and polymorphism, enabled by the use Java, that allows the user to make re-usable and extendable code.

2.3 Modular

Tandem allows the user to write modular programs. Every node represents a separate function and contains all of the information needed to execute that function. This allows for compartmentalization and separation of concerns. This makes code in Tandem very reusable and allows for large projects to be broken down into more manageable sub-projects where a team working on one sub-project needs to know very little about what other teams are working on.

Additionally, developers can create libraries by combining nodes to form super nodes, and users can use these libraries by considering the inputs each super node takes and possible outputs that it can produce. The end result is that users can use modules written by others without worrying about the minute manipulations going on inside an individual state.

2.4 Immutable

Tandem heavily emphasizes immutability. All data in a node is encapsulated within that node, which means nodes cannot share any state. This requires a shift in the programmer's mindset, but allows a developer to not have to worry about other nodes affecting the node at hand. While variables internal to a node may be re-assigned, Tandem does not let users create global variables that can later be manipulated. This means that the programmer does not need to worry about mutability changing any data structures and instead can focus on all possible inputs that may be received and any possible outputs that should be sent to the next state.

2.5 Massively Parallel

Tandem programs allow for massively parallel programs written naturally. As systems are developed with more and more cores, it becomes important to be able to program on each of these cores without having to worry about threads and locks, and the parallel nature of Tandem allows for this. Multiple transitions are allowed for the same output, with non-determinism being simulated by Java threads. Nodes can quickly branch out to simulate all the different possibilities for an algorithm, and immutability means that nodes do not have to worry about inadvertently creating race conflicts or altering some shared mutable state. The ease with which Tandem will run these concurrent and completely separate processes on a system makes the language support massively concurrent algorithms, and in fact makes programs embarrassingly parallel. Finally, because each thread can run at the same time with its own copies of data, and because there is no communication between the processes, Tandem allows for the fast and safe execution of programs.

3 Practical Aspects of Tandem

3.1 Compiled

Since Tandem is built using Java, it is compiled into bytecode, which in turn is interpreted by the Java Virtual Machine (JVM). Compiling to the JVM enables the user to attain high performance for their code, not have to worry about allocating or freeing memory, and having a stable and secure environment on which Tandem code can run. However, this does create the drawback that users will need to compile their results before they can see any speed benefits, and takes away the high turnover-rate of modern dynamic languages like Python or

Ruby, where code can be run and tested quickly without having to wait for it to compile first. The inclusion of an interpreter later on can help mitigate this cost.

3.2 Dynamic Type System

Tandem is a dynamic programming language. While Tandem retains a sense of types so that it can compile to Java bytecode, it does not force the user to specify types statically, like in C or Java, nor do users have to worry about fixing type errors simply to satisfy the compiler. While this does introduce some complexity for us, the compiler's writers, users gain the benefit of being able to focus only on the code and nothing extraneous to the task at hand.

3.3 Architecture-Independent

Tandem is architecture-independent. The language is compiled to run upon the Java Virtual Machine, and thus inherits the benefits of the JVM, including architecture-independence and portability. An architecture-independent language runs the same regardless of the type of assembly instructions that an individual machine supports, and runs the same on 32-bit and 64-bit systems. Architecture-independent languages also have the benefit of being easy to port across multiple operating systems, so a programmer who writes on a Linux machine will have no problem debugging or maintaining the code on Windows or OS X. If a machine supports the JVM, then Tandem can run on that machine.

3.4 Threaded

Because Tandem consists of nodes that can transition to multiple states, we can make use of Javas multi-threaded capabilities with the Thread class to simulate the various transitions that can be made, especially to simulate nondeterminism. The functionality of the class allows for threads to run concurrently without disturbing each other, which makes processes safe and stable as long as individual threads do not alter shared state. Because each node in Tandem is immutable, as a node in a thread transitions and executes some function, it operates on its own working copy of a variable. The language puts restrictions on what shared state, if any, can be accessed from within an individual node and lets us create as many threads as necessary to simulate states without worrying about corrupted memory or race-conditions.