Poster Abstract: PUSH, a Dataflow Shell

Noah Evans
Alcatel-Lucent Bell Labs
Antwerp, Belgium
npe@plan9.cs.bell-labs.com

Eric Van Hensbergen
IBM Research
Austin, TX
bergevan@us.ibm.com

The deluge of huge data sets such as those provided by sensor networks, online transactions, and the web provide exciting opportunities for data analytics. The scale of the data makes it impossible to process in a reasonable amount of time on isolated machines. This has lead to data flow systems emerging as the standard tool for solving research problems using these vast datasets. In typical dataflow systems, runtimes [2] [1] [4] [**?**] and define graphs of processes, the edges of the graphs representing pipes and their vertices representing computation on a system. Within these runtimes a new class of languages [7] [8] [6] can be used by researchers to solve "pleasantly parallel" problems(problems where the individual elements of datasets are considered to be independent of any other element) more quickly without worrying about explicit concurrency.

These languages provide automated control flow(typically matched to the architecture of the underlying runtime) and channels of communication between systems. In existing systems, these workflows and the underlying computation are tightly linked, tying solutions to a particular runtime, workflow and language. This creates difficulties for analytics researchers who wish to draw upon tools written in many different languages or runtimes which may be available on several different architectures or operating systems.

We observe that UNIX pipes were designed to get around many of these incompatibilities, allowing developers to hook together tools written in different languages and runtimes in ad-hoc fashions. This allowed tool developers to focus on doing one thing well, and enabled code portability and reuse in ways not originally conceived by the tool authors. The UNIX shell incorporated a model for tersely composing these smaller tools in pipelines (e.g. 'sort | uniq -c'), creating coherent workflow to solve more complicated problems quickly. Tools read from standard input and wrote to standard output, allowing programs to work together in streams *with no explicit knowledge of this chaining built into the program itself.*

One to one pipelines such as those used by a typical UNIX shell, can not be trivially mapped to streaming workflows which incorporate one-to-many, many-to-many, and many-to-one data flows. Additionally, typical UNIX pipeline tools write data according to buffer boundaries instead of record boundaries. As [7] notes dataflow systems need to be able to cleanly separate input streams into records and then show that the order of these records is independent. By separating input and output into discrete unordered records data can be easily distributed and coalesced.

To address these issues we have implemented a prototype shell, which we call PUSH, using dataflow principles and incorporating extended pipeline operators to establish distributed workflows —potentially running on clusters of machines— and correlate results. Our implementation is based on extending an existing shell, MASH[3], from which we inherited a rich interpreted scripting language. It treats variables as lists of strings and has no native handling for any other data type. Integer expression handling and other facilities are provided by shell commands. It has native regular expression support and it has a novel ability to do declarative shell programming through a make like syntax incorporated in the shell itself.

We currently have a working prototype of the PUSH shell, which we can use to target local distributed clusters, dynamic clusters built using Amazon's EC2 cloud, and large scale clusters such as a Blue Gene running the kittyhawk infrastructure. We are currently in the process of evaluating and optimizing performance for a variety of application types.

We are also implementing a new version of Push, this one based on the RC shell[**?**] to easier integration into traditional unix systems like Linux. This version is simplified and closer to the bourne shell. The explicit goal of the new version of Push is integration with the Unified Execution Model(UEM)[**?**] which will allow the transparent distribution of processes and the connection of their communication channels between machines transparently. This work is taking place as part of the HARE project[**?**]

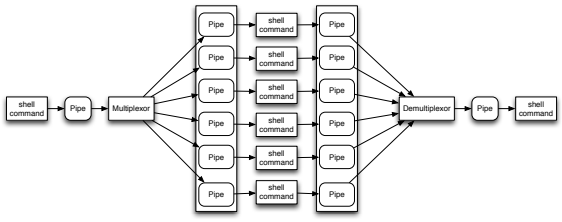

Figure 1: The structure of the PUSH shell

We have added two additional pipeline operators, a multiplexing fan-out(|<[n]), and a coalescing fan-in(>|). This combination allows PUSH to distribute I/O to and from multiple simultaneous threads of control. The fan-out argument n specifies the desired degree of parallel threading.

Eurosys April 14–16, 2010, Paris, France.
ACM X-XXXXX-XX-X/XX/XX.

If no argument is specified, the default of spawning a new thread per record (up to the limit of available cores) is used. This can also be overriden by command line options or environment variables. The pipeline operators provide implicit grouping semantics allowing natural nesting and composibility. While their complimentary nature usually lead to symmetric mappings (where the number of fan-outs equal the number of fan-ins), there is nothing within our implementation which enforces it. Normal redirections as well as application specific sources and sinks can provide alternate data paths. Remote thread distribution and interconnect are composed and managed using synthetic file systems in much the same manner as Xcpu,[5] pushing the distributed complexity into the middleware in an language and runtime neutral fashion.

PUSH also differs from traditional shells by implementing native support for record based input handling over pipelines. This facility is similar to the argument field separators, IFS and OFS, in traditional shells which use a pattern to determine how to tokenize arguments. PUSH provides two variables, ORS and IRS, which point to record separator modules. These modules (called multiplexors in PUSH) split data on record boundaries, emitting individual records that the system distributes and coalesces.

The choice of which *multipipe*, an ordered set of pipes, to target is left as a decision to the module. Different data formats may have different output requirements. Demultiplexing from a multipipe is performed by creating a many to one communications channel within the shell. The shell creates a reader processes which connects to each pipe in the multipipe. When the data reaches an appropriate record boundary a buffer is passed from the reader to the shell which then writes each record buffer to the output pipeline.

An example from our particular experience, Natural Language Processing, is to apply an analyzer to a large set of files, a "corpus". User programs go through each file which contain a list of sentences, one sentence per line. They then tokenize the sentence into words, finding the part of speech and morphology of the words that make up the sentence. This sort of task maps very well to the DISC model. There are a large number of discrete sets of data whose order is not necessarily important. We need to perform a computationally intensive task on each of the sentences, which are small, discrete records and ideal target for parallelization.

PUSH was designed to exploit this mapping. For example, to get a histogram of the distribution of Japanese words from a set of documents using chasen, a Japanese morphological analyzer, we take a set of files containing sentences and then distribute them to a cluster of machines on our network. The command is as follows:

```
push -c '{
  ORS=./blm.dis
  du -an files |< xargs os \\
   chasen | awk '{print \$1}' | sort | uniq -c \\
   >| sort -rn
}'
```

The first variable, ORS, declares our record multiplexor module, the intermediary used to ensure that the input and output to distributed pipes are correctly aligned to record boundaries. du -n gives a list of the files (note that our du is a bit different from the canonical UNIX du, it replaces much of find's functionality) which are then "fanned out"(`|<`) using a combination of a multipipes, and a *multiplexor* which determines which pipes are the targets of each unit of output. This fanned out data goes to xargs on other machines which then uses the filenames(sent from the instantiating machine) as arguments to chasen. The du acts as a command driver, fanning out file names to the individual worker machines. The workers then use the filenames input to xargs, which uses the input filenames as arguments to xargs target command. Using the output of the analyzer awk extracts the first line fields(Japanese words) which are then sorted and counted using uniq. Finally these word counts are "fanned in"(`>|`) to the originating machine which then sorts them.

PUSH is under active development, but the current source code is open source and available on Google Code. This work has been supported by the Department of Energy Of Office of Science Operating and Runtime Systems for Extreme Scale Scientific Computation project under contract #DE-FG02-08ER25851.

1. ACKNOWLEDGEMENTS

Willem De Bruin and Pascal Wolkotte provided extensive feedback on this paper. Their help is greatly appreciated.

2. REFERENCES

[1] A. Bialecki, M. Cafarella, D. Cutting, and O. O Malley. Hadoop: a framework for running applications on large clusters built of commodity hardware, 2005. *Wiki at http://lucene. apache. org/hadoop*.

[2] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(01):7, 2008.

[3] Bruce Ellis. Mash Manual Page. *Vita Nuova Ltd http://www .vitanuova .com/inferno/man/1/mash .html*.

[4] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2007 conference on EuroSys*, pages 59–72. ACM Press New York, NY, USA, 2007.

[5] Ron Minnich and Andrey Mirtchovski. Xcpu: a new, 9p-based, process management system for clusters and grids. *Cluster Computing, 2006 IEEE International Conference on*, pages 1–10, 2006.

[6] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM New York, NY, USA, 2008.

[7] R. Pike. Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming*, 13(4):277–298, 2005.

[8] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P.K. Gunda, and J. Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *Symposium on Operating System Design and Implementation (OSDI), San Diego, CA, December*, pages 8–10, 2008.