PaSh: Light-Touch Data-Parallel Shell Processing

Nikos Vasilakis*  Konstantinos Kallas*  Konstantinos Mamouras
MIT  University of Pennsylvania  Rice University

Achilles Benetopoulos  Lazar Cvetković
(Unaffiliated)  University of Belgrade

* equal contribution
Shell Scripts are Everywhere

Default/scriptable system interface even in the lightest containers
Kubernetes, Docker

Universal composition environment
Commands (programs) can be written in C, C++, Rust, JS, Python, Ruby, Haskell...

Succinct data processing:
download/extraction/preprocessing/querying

```bash
# Check all possible clusters, as your .kubeconfig may have multiple contexts:
kubectl config view -o jsonpath='{"Cluster name\nServer\n"}{range .clusters[*]}{

# Select name of cluster you want to interact with from above output:
export CLUSTER_NAME="gke_ps-dev-201405_us-east1_acaterberg"
export SERVICE_ACCOUNT=joc
export SERVICE_ACCOUNT=cloudbes-core-nginx-ingress
export SERVICE_ACCOUNT=default

# Point to the API server referring the cluster name
APISERVER=$(kubectl config view -o jsonpath="{.clusters[?(@.name="$CLUSTER_NAME")].apiServer}")

echo "Building parser..."
cd compiler/parser
echo "|-- making libdash..."
makedb && $LOG_DIR/make_libdash.log

seq $FROM $TO | sed 's;\^;\$IN;' | sed 's;\$;\n;/' | xargs -r -n 1 $fetch | grep gz | tr -s ' \n' | cut -d ' ' -f9 | sed 's;\^\(\[20-0-9]\[8-9]\).gz;\2/\1\2.gz;g' | sed 's;\^;\$IN;' | xargs -n1 curl -s | gunzip |

# end-preprocessing

cut -c 89-92 | grep -v 999 | sort -rn | head -n1  # actual processing
```
A Classic Shell Script

Bentley: A word-counting challenge

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair.

10 was
10 the
10 of
10 it
2 times

Knuth: 100s of lines of literate WEB

Mcllroy: Unix one-liner

tr -cs A-Za-z \n | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair.

```
tr -cs A-Za-z \n | tr A-Z a-z | sort | uniq -c | sort -rn | sed \${1}q
```
It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair.
It was the best of times it was the...

tr A-Z a-z

it was the best of times it was the...
it
was
the
best
of	
times
it
was
the
...

sort

age
age
belief
best
darkness
despair
epoch
epoch
foolishness
...

tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
```
tr -cs A-Za-z \n | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
```
tr -cs A-Za-z '
' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
tr -cs A-Za-z \n | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of...

How to parallelize?

tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q
Shell scripts are mostly sequential

Their parallelization requires considerable effort:

- Command-specific flags (e.g., `sort -p`, `make -jN`)
- Mostly-manual, restricted parallelization tools (e.g., GNU `parallel`)
- Full rewrites in parallel frameworks (e.g., MapReduce)
public class top_10_Movies_Mapper extends Mapper<Object, Text, LongWritable, Text> {
    private TreeMap<Long, String> tmap;

    @Override
    public void setup(Context context) throws IOException, InterruptedException {
        tmap = new TreeMap<Long, String>();
    }

    @Override
    public void map(Object key, Text value, Context context)
                    throws IOException, InterruptedException {
        // no_of_views (tab seperated)
        // we split the input data
        String[] tokens = value.toString().split("\t");
        String movie_name = tokens[8];
        long no_of_views = Long.parseLong(tokens[1]);
        tmap.put(no_of_views, movie_name);
        if (tmap.size() > 10) {
            tmap.remove(tmap.firstKey());
        }
    }

    @Override
    public void cleanup(Context context) throws IOException, InterruptedException {
        for (Map.Entry<Long, String> entry : tmap.entrySet()) {
            long count = entry.getKey();
            String name = entry.getValue();
            context.write(new Text(name), new LongWritable(count));
        }
    }
}

import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.hadoop.mapreduce.Mapper;

public class top_10_Movies_Reducer extends Reducer<Text, LongWritable, Text, LongWritable> {
    private TreeMap<Long, String> tmap2;

    @Override
    public void setup(Context context) throws IOException, InterruptedException {
        tmap2 = new TreeMap<Long, String>();
    }

    @Override
    public void reduce(Text key, Iterable<LongWritable> values, Context context)
                        throws IOException, InterruptedException {
        String name = key.toString();
        long count = 0;
        for (LongWritable val : values) {
            count += val.get();
        }
        tmap2.put(count, name);
        if (tmap2.size() > 10) {
            tmap2.remove(tmap2.firstKey());
        }
    }

    @Override
    public void cleanup(Context context) throws IOException, InterruptedException {
        for (Map.Entry<Long, String> entry : tmap2.entrySet()) {
            long count = entry.getKey();
            String name = entry.getValue();
            context.write(new LongWritable(count), new Text(name));
        }
    }
}

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class Driver {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        if (otherArgs.length < 2) {
            System.err.println("Error: please provide two paths");
            System.exit(2);
        }

        Job job = Job.getInstance(conf, "top 10");
        job.setJarByClass(Driver.class);
        job.setJarPathBase(new Path(otherArgs[0]).toUri().getPath());
        job.setInputFormatClass(FileInputFormat.class);
        job.setOutputFormatClass(FileOutputFormat.class);
        job.setMapperClass(top_10_Movies_Mapper.class);
        job.setReducerClass(top_10_Movies_Reducer.class);
        job.setMapOutputKeyClass(LongWritable.class);
        job.setMapOutputValueClass(Text.class);
        job.setOutputKeyClass(LongWritable.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

Big-Data Version
of McIlroy’s Pipeline
150-line Hadoop Program
Parallelization requires considerable effort:

- Command-specific flags (e.g., `sort -p`, `make -jN`)
- Mostly-manual, restricted parallelization tools (e.g., GNU `parallel`)
- Full rewrites in parallel frameworks (e.g., MapReduce)

Mostly sequential by default — how to parallelize?
Challenges of Automating Shell-Script Parallelization

for directory in /project/gutenberg/*/; do
  ls $directory | grep 'txt' | wc -l > index.txt
done

cat f1 f2 | tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c | sort -rn | sed ${1}q

echo 'Done';

(1) Numerous and opaque Unix commands
(2) Shell language enforced dependencies
(3) Runtime support for Unix parallelization
# PaSh Overview

## Compile

```bash
cat $f1 f2 | sort
```

## Parse

```bash
cat f1 f2
sort
```

## Optimize

```
cat f1 f2 | sort
```

## Annotations

```
Optimized DFG
```

## RunTime Library

```bash
mkfifo a b
sort f1 > a &
sort f2 > b &
sort -m a b &
wait;rm -f a b
```

## Emit

```
f1
sort
f2
sort
sort -m
```

## Unparse

```
AST
mkfifo a b
&
>

sort f1
a
```
PaSh Overview

**seq.sh**

```sh
cat $f1 $f2 | sort
```

**par.sh**

```sh
mkfifo a b
sort $f1 > a &
sort $f2 > b &
sort -m a b &
wait;rm -f a b
```

**Compile**

```sh
cat $f1 $f2 | sort
```

**Optimize**

```sh
sort $f1 > a &
sort $f2 > b &
sort -m a b &
wait;rm -f a b
```

**Unparse**

**Emit**

**Annotations**

**Runtime Library**

**DFG**
1. Unix Parallelizability Study & Annotations
Parallelizability properties:
- 4 broad classes
- Flags and options
- Input consumption

Parallelizability DSL:
(cmd, flg, [in]) → DFG node
command parallelizability classes

12.7% stateless

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair.
It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair.
command
parallelizability
classes

12.7%  stateless
8.7%   parallelizable pure
8.2%   non-parallelizable pure

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair.

input.txt

+state

sha1shum

+state

sha1shum

x
command parallelizability classes

12.7% stateless
8.7% parallelizable pure
8.2% non-parallelizable pure
70.4% side-effectful
PaSh Overview

seq.sh

cat $f1 f2 | sort

pars.sh

mkfifo a b
sort f1 > a &
sort f2 > b &
sort -m a b &
wait;rm -f a b

Annotations

Runtime Library

Emit
PaSh Overview

**Compile**

```
seq.sh
```

```
cat $f1 f2 | sort
```

**Parse**

```
par.sh
```

```
mkfifo a b
sort f1 > a &
sort f2 > b &
sort -m a b &
wait;rm -f a b
```

```
AST

cat f1 f2 ->
sort

AST

mkfifo a b &

sort f1 a
```

**Unparse**

```
AST

| |
```

```
AST

cat f1 f2 ->
sort
```

**Compile**

```
AST

cat f1 f2 ->
sort
```

**Annotations**

```
Optimize
```

```
AST

mkfifo a b &

sort f1 a
```

**Runtime Library**

```
AST

mkfifo a b &

sort f1 a
```

```
mt -m a b &

wait;rm -f a b
```

```
Optimize
```

**Emit**

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```

```
Optimized DFG
```
2. Dataflow Model & Transformations
cat f1 f2 > out.txt; cat out.txt
cat f1 f2 | tr A-Z a-z | sort > out.txt
cat f1 f2 | tr A-Z a-z | sort > out.txt

Transformation condition: tr is stateless
cat f1 f2 | tr A-Z a-z | sort > out.txt

Transformation condition: cat followed by split
cat f1 f2 | tr A-Z a-z | sort > out.txt

Transformation condition: sort is parallelizable pure
Transformation condition: cat followed by split
1 + 3 Transformations
PaSh Overview

**seq.sh**
```
cat $f1 f2 |
sort
```

**parse.sh**
```
AST
```
```
cat f1 f2 >
sort
```

**Compile**
```
AST
```
```
cat f1 f2 >
sort
```

**Annotations**
```
f1
```
```
f2
```
```
sort
```
```
sort -m
```

**Runtime Library**
```
par.sh
```
```
mkfifo a b
sort f1 > a &
sort f2 > b &
sort -m a b &
wait;rm -f a b
```

**Unparse**
```
AST
```
```
; ;
```
```
mkfifo a b &
```
```
>
```
```
sort f1 a
```

**Emit**
```
`Optimized DFG`
```
```
Optimized DFG
```

**Optimize**
```
DFG
```
```
f1
```
```
f2
```
```
sort
```
```
sort
```
```
sort -m
```
```
Optimized DFG
```
PaSh Overview

**seq.sh**

```bash
cat $f1 f2 | sort
```

**ParSh**

```bash
mkfifo a b
sort f1 > a &
sort f2 > b &
sort -m a b &
wait;rm -f a b
```

**Compile**

```
cat $f1 f2 | sort
cat f1 f2 > |
```

**Optimize**

```
| sort
```

**Unparse**

```
cat
```

**Annotations**

```
> 
```

**Emit**

```
sort f1 > a
```

**Runtime Library**

3
3. Runtime Support
Runtime Support: Performance & Correctness

- Unix pipes are lazy, i.e., inadequate buffering (and for a good reason)
- Dataflow graph termination is tricky
- Parallelizable-pure commands require careful aggregation
Runtime Challenge: Unix's Lazy Semantics

```bash
mkfifo f1 f2
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
cat f1 f2
```
Runtime Challenge: Unix's Lazy Semantics

```
mkfifo f1 f2
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
cat f1 f2
```
Runtime Challenge: Unix's Lazy Semantics

mkfifo f1 f2

grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
cat f1 f2
Runtime Challenge: Unix's Lazy Semantics

```
mkfifo f1 f2
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
cat f1 f2
```
Runtime Challenge: Unix's Lazy Semantics

mkfifo f1 f2
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
cat f1 f2
Runtime Challenge: Unix's Lazy Semantics

```sh
mkfifo f1 f2
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
cat f1 f2
```

Execution proceeds in steps!
Among other problems, this "solution" prevents pipeline parallelism (more on that later)
The PaSh Solution: Eager Buffers

```bash
mkfifo f1 f2 f3 f4
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
eager < f1 > f3 &
eager < f2 > f4 &
cat f3 f4
```
The PaSh Solution: Eager Buffers

```bash
mkfifo f1 f2 f3 f4
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
eager < f1 > f3 &
eager < f2 > f4 &
cat f3 f4
```
The PaSh Solution: Eager Buffers

```
mkfifo f1 f2 f3 f4
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
eager < f1 > f3 &
eager < f2 > f4 &
cat f3 f4
```
The PaSh Solution: Eager Buffers

mkfifo f1 f2 f3 f4
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
eager < f1 > f3 &
eager < f2 > f4 &
cat f3 f4
The PaSh Solution: Eager Buffers

```bash
mkfifo f1 f2 f3 f4
grep "foo" in1 > f1 &
grep "foo" in2 > f2 &
eager < f1 > f3 &
eager < f2 > f4 &
cat f3 f4
```

/pash/runtime/eager

- Unix command, usable outside PaSh too
- Buffers input eagerly – can spill to disk
- Keeps fragment in DFG model
Demo Time!
Evaluation
1. Expert / Classic Scripts

Speedups against bash baseline for pash --width=16:

5.93x vs. 8.83x
2. Pipelines in the wild

+ PaSh awareness goes a long way!

```
e.g. #26

{ cat $IN6 | awk '{print $2, $0}' | sort -nr | cut -d ' ' -f 2 (1.01x) 
  cat $IN6 | sort -nr -k2 | cut -d ' ' -f 1 (8.1x !!1!1) }
```

Configuration:
- Full PaSh
  --width=16
### 3. Case Study no.1: NOAA Weather Analysis

**fetch, preprocess, cleanup, filter**

<table>
<thead>
<tr>
<th>Command</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bash</td>
<td>33m58s</td>
</tr>
<tr>
<td><code>pash -w 16</code></td>
<td>16m39s</td>
</tr>
</tbody>
</table>

**calculate**

<table>
<thead>
<tr>
<th>Command</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bash</td>
<td>10m4s</td>
</tr>
<tr>
<td><code>pash -w 16</code></td>
<td>49s</td>
</tr>
</tbody>
</table>

- **2.04x** speedup for preprocessing
- **12.31x** speedup for preprocessing
- **2.52x** combined speedup for the full program

Hadoop only focuses on this part.

This part is not the focus of traditional parallelization frameworks but parallelizing it has the biggest impact.

**Configuration:**
- Full PaSh
- `--width=16`
- 82GB (5y data)
Conclusion
Conclusion

- Parallelize unix shell scripts (POSIX -> POSIX)
- Annotations address extensibility issues
- Open source — 12+ contributors
- Lots of recent excitement — let's rehabilitate the shell!

pash-discuss@googlegroups.com
github.com/andromeda/pash