Using AWK and R to parse 25TB (livefreeordichotomize.com)

312 points by markus_zhang on June 27, 2019 | hide | past | favorite | 104 comments

stirfrykitty on June 27, 2019 | next [–]

There was a similar article (2014) that is also interesting. I think too many of us see new and shiny and immediately glom onto it, forgetting that the UNIX/regex fathers knew a thing or two about crunching data.

https://adamdrake.com/command-line-tools-can-be-235x-faster-...

close04 on June 27, 2019 | parent | next [–]

> often people use Hadoop and other so-called Big Data ™ tools for real-world processing and analysis jobs that can be done faster with simpler tools and different techniques.

Right tool for the right job, as always. For a 2-3GB dataset size you don't need to bother with Hadoop just as for a 2-3PB dataset size you probably don't need to bother with awk.

taeric on June 27, 2019 | root | parent | next [–]

If like to think that it is feasible most 2-3PB datasets can be easily partitioned to GB datasets. I rather guess it is more common to expand GB datasets into PB ones, though. :(

malshe on June 27, 2019 | parent | prev | next [–]

Thanks for the link. Very interesting.

In OP's article there is a link to a book "Data Science at the Command Line" which sounds quite relevant:
https://www.datascienceatthecommandline.com

markus_zhang on June 27, 2019 | root | parent | next [–]

Yeah I decide to read that book and install the Docker file. The thing is I'm kind of not sure how to setup the whole thing in Linux as I also need Python, some database and gcc. I think I should be able to find some tutorials for Ubuntu.

jeroenjanssens on June 27, 2019 | root | parent | next [–]

Although the docker image is based on Alpine Linux, examining the corresponding Dockerfile [1] may provide some guidance on how to install the tools and their requirements on Ubuntu. Let me know if you have any questions. Always happy to help.

[1] https://github.com/datascienceworkshops/dockerfiles/blob/mas...

rout39574 on June 27, 2019 | parent | prev | next [–]

You might be on to something there.

samuell on June 27, 2019 | prev | next [–]

I counted to 15 awk calls in our latest pipeline processing drug compounds to build predictive machine learning models of them:
https://github.com/pharmbio/ptp-project/blob/master/exp/2018...

One of the most eye-opening aspects of awk (goes for other pipeable commandline tools too), was how they support iterative development of pipelines regardless of data size.

We tried SQLite at some point for some of the stages because of pretty complicated selections sometimes, but it often won't give back a single result in minutes. Switching to AWK, I could immediately get *some* output, so I could quickly validate and iterate on the awk code until I got what I expected. The actual execution was likeways always very fast.

olodus on June 27, 2019 | prev | next [–]

Awk is kinda my new favorite scripting language. I've been using it for the master's thesis I'm writing and am amazed at how quickly I can script in it. And how easy it was to learn. From the beginning out results-visualisation pipeline of awk+gnuplot was just supposed to be a quick hack to get something we could look at but it has kept up the whole thesis through and just been extended and made better instead of switching. We still use python when we need some lib help to get some data right but damn it goes quick to handle well structured data with awk. Sad I didn't learn it earlier.

et2o on June 27, 2019 | prev | next [–]

My biggest question is why on earth did they give you 25 Tb of TSV genetic data? :-)

I'm not sure what your sample was but seems like it would have been better to use one of the special binary file formats for genetic data. You wrote SNP chips, But in order to get to 25 Tb I assume there must be imputed calls, so it seems like a BGEN might have been a lot easier.

This is speculation of course, I'm not sure exactly what your situation was.

chrchang523 on June 27, 2019 | parent | next [–]

You're correct that, for the final discrete or probabilistic variant calls, there are far better data formats. However, it's clear that Nick's lab currently wants to work with raw intensity readings.

My main practical recommendation for Nick is to become familiar with bgzip and zstd. bgzip sacrifices a little bit of compression efficiency relative to plain gzip, but in exchange it solves the more important problems of (i) letting you take advantage of all your cores when decompressing and (ii) supporting random-access reads with an appropriate index, while remaining compatible with all .gz-reading programs. When backward compatibility is unimportant, zstd tends to have much better compression/decompression speed for the same compression ratio than gzip.

wongarsu on June 27, 2019 | parent | prev | next [–]

If you unpack all of https://files.pushshift.io/reddit/comments/ you have many Tb of JSONs that are just dumps of API responses that slowly change schema over the years. It's also an incredibly useful dataset.

In the end CPUs are fast enough and compression algorithms good enough that I would argue it doesn't really matter what format you use for storage, as long as it's reasonably easy to read back.

et2o on June 27, 2019 | root | parent | next [–]

In the case of genomics, there have been at this point decades of work developing high performance file formats and there are large ecosystems of tools around them. Lots of bioinformatics is really manipulating these files. So using a supported file format makes a big difference.

nstrayer on June 27, 2019 | parent | prev | next [–]

It's a plain chip, just one specially made for our institution by illumina. As to why they would deliver it in tsvs, that I can't answer.

nstrayer on June 27, 2019 | prev | next [–]

Hi! Author of the post here. I can attempt to answer any questions if need be although it looks like others have done a great job doing that already!

markus_zhang on June 27, 2019 | parent | next [–]

Hi nstrayer thanks for the excellent article!

I work as a data analyst and I never got to worry about big data as the DWH takes care of the aggregation for us, plus I only work in Windows.

I see now that it would be very useful to learn *nix tools in general, as it seems that the skills to process (not to predict/analyze) terabytes+ data are very valuable and expensive to acquire and could be one's butter and bread.

banku_brougham on June 29, 2019 | root | parent | next [–]

Windows OS is making it easier to access nix terminal commands, but in my experience making the OS switch (mac machine but unix terminal, or cloud linux) has been game changing for me.

heuermh on June 27, 2019 | parent | prev | next [–]

Hello! I'm wondering if you came across our suite of libraries and tools for doing Genomics on Spark?

https://github.com/bigdatagenomics/adam

We've fallen off the first Google hit the past few years but are still quite relevant (e.g. Databricks' commercial offering uses ADAM under the hood). Drop in our Gitter some time!

jcastro on June 27, 2019 | parent | prev | next [–]

Love your writing style, I'm not anywhere close to working in this field and I learned something.

nstrayer on June 27, 2019 | prev | next [–]

As a followup to this. We have now successfully run complex statistical models across all 2.5 million snps on a single AWS instance in less than 3 hours just by writing R code using the package I describe at the end of the article.

wikibob on June 27, 2019 | parent | next [–]

Have you considered using AWS EC2 Spot Instances? The price can be 50 to 70% cheaper.

You would have to add some additional reliability to your pipeline so it could continue processing when instances are unexpectedly terminated. But this might be well worth it as it sounds like your research group is cost-constrained.

AWS made some changes this year so the spot prices are more stable and instances don't get shut down as frequently.

nstrayer on June 27, 2019 | root | parent | next [–]

I did use spot instances for most of the clusters and a few of the processing jobs! I got out of the habit of using them earlier due to loosing them but now that they have the 'pay up to the on-demand price' option they're great!

kermatt on June 27, 2019 | prev | next [–]

Have used Mawk [1] in similar cases for a runtime savings, provided that the script works without any GNU Awk extensions.

[1] https://invisible-island.net/mawk/

mzs on June 27, 2019 | parent | next [–]

Thank you! For those that don't know, 15 years ago GNU awk was sometimes oddly REALLY slow. Mike's awk was not. Plus when things were overall slower back then, it mattered more.

But there were bugs in mawk and it seemed basically unmaintained. So you'd run into something and have to use gawk or perl instead.

That's no longer the case, the xterm guy adopted it, ten years ago, and now I know!

kermatt on June 27, 2019 | root | parent | next [–]

Last release was late 2017, but it has been very stable for me. Plus the author responds to bug reports.

Mawk is my go-to version because of speed. GNU Awk when its extensions are needed, or the task is over "small data" and the system default version is sufficient.

IndustrialJane on June 27, 2019 | parent | prev | next [–]

I always wondered why Mawk did not implement Gawk's extensions - or at least those that could be done without any major penalty for the rest of the code.

Do you know why?

geogra4 on June 27, 2019 | prev | next [–]

I remember frantically needing to parse a few hundred megs of log files and join them up to the db rows that fired the individual errors.

Initially I was trying to use SQLite for it but I kept running out of memory and crashing the system. Turned out using grep, join, sort, and paste got the job done in seconds.

nimrody on June 27, 2019 | prev | next [–]

Very well written.

I think using 'make' with the -j parameter (# of parallel jobs) is more useful than using gnu 'parallel'. The reason is that if one of the job fails for some reason, you just re-run 'make' and only the required jobs are started instead of restarting the entire computation.

IndustrialJane on June 27, 2019 | parent | next [–]

With --results or --joblog and --resume-failed GNU Parallel can do this, too.

innomatics on June 27, 2019 | prev | next [–]

Great post and thanks for sharing your learnings.

A couple of quick questions:

Was the 25TB raw data gathered from a single human genome?

What would be the size in bytes of a unique genomic fingerprint once raw data is all fully processed into high confidence base values? (including non-coding regions)

If we just look at coding regions and further compress by only looking at SNPs, how many bytes is that?

Considering that each base has ~2B of information... it would be super interesting to know how much space it takes to describe our uniqueness!

eesmith on June 27, 2019 | parent | next [–]

Q: "Was the 25TB raw data gathered from a single human genome?"

Essay: "Each row contained a data for a single SNP for a single person." ... "There were ~2.5 million SNPS and ~60 thousand people"

Statement: "it would be super interesting to know how much space it takes to describe our uniqueness"

Genome size is "3,234.83 Mb (Mega-basepairs) per haploid genome" says https://en.wikipedia.org/wiki/Human_genome .

However, question of *uniqueness* depends on your model. We are all unique.

If you have human genomes, then use a reference template and only list those which differ. Eg, https://en.wikipedia.org/wiki/Compression_of_Genomic_Sequenc... .

sszz on June 27, 2019 | parent | prev | next [–]

In order to uniquely identify, not a lot of space! A recent paper puts it at 50 SNPs (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5785835/).

Describing your total unique genetic profile would obviously require a lot more space, and wouldn't be constant across individuals/ancestral backgrounds (e.g. there's more genetic diversity in people of African descent).

markus_zhang on June 27, 2019 | parent | prev | next [–]

Sorry I saw this article and thought it was pretty interesting. This is NOT my article but I'd like to know what others would do under this situation.

BTW not sure, but is it OK to post other's article here? Maybe I should add a short commentary in the title.

kohtatsu on June 27, 2019 | root | parent | next [–]

The default assumption is that it's not your article, unless you prepend "Show HN" (or there's something obvious like your username matching the domain name).

nstrayer on June 27, 2019 | root | parent | prev | next [–]

Of course! Glad you posted it. I didn't realize that many people would find it interesting.

innomatics on June 27, 2019 | root | parent | prev | next [–]

I think it's fine. The author might show up otherwise others with interest in the area might reply.

kreetx on June 27, 2019 | root | parent | prev | next [–]

You don't even need to say it's not yours, probably most that is posted here isn't from the author.

usgroup on June 27, 2019 | prev | next [–]

Lol , this is basically how I roll most the time . However what Linux is really missing right now is command line tools that saturate a GPU.

Just counterparts to all the favourites that utilise the GPU ... imagine GPU awk.

_hl_ on June 27, 2019 | parent | next [–]

> imagine GPU awk

My intuition tells me that awk and other text processing tools won't scale well to a GPGPU. I might be wrong though. Is there any example of something like grep etc working well on a GPU?

usgroup on June 27, 2019 | root | parent | next [–]

Pffff naysayers ...

https://www.cs.cmu.edu/afs/cs/academic/class/15418-s12/www/c...

_hl_ on June 27, 2019 | root | parent | next [–]

That's very surprising, assuming they didn't doctor the results by choosing the workload all too carefully.

I would have expected a GPU regex too perform much worse, given that regex matching is probably very branchy code. Especially since computation is generally way faster than IO.

kuzehanka on June 28, 2019 | root | parent | next [–]

Modern GPU have no issues with branching.

reitzensteinm on June 28, 2019 | root | parent | next [–]

What specifically are you referring to? Branching on GPUs has not substantially changed for a decade. If all threads on warp skip a branch, it's free. If one takes it, the rest also pay the penalty and mask out the vector units.

What's at play here is that the needle in a haystack search of regex is going to spend almost all its time 0 or 1 deep in the state machine, so the threads skip the branches and the penalty is not large.

KingMachiavelli on June 27, 2019 | root | parent | prev | next [–]

When this concept was previously posted on HN, the top comment pointed out how it's the *pipes* that are inefficient when working on GPUs due to copying data from the CPU to the GPU and vice-versa for each command & pipe pair. I think even if we don't get pipes per se but I think we could expose GPGPU resources in a unix-like way but I suppose it depends on driver support.

But as to your question, a lot of traditional tools like grep, sed, aren't really suited for the GPU unless you are running them on a lot of files at once.

https://news.ycombinator.com/item?id=5803943

superdimwit on June 27, 2019 | root | parent | prev | next [–]

I agree. I get the sense that a lot of these old text munging tools are seriously fast, and are mostly IO-bound. It would surprise me if moving chunks of text back and forth to a GPU would be very efficient.

ulrikrasmussen on June 27, 2019 | root | parent | prev | next [–]

There is at least some work on data-parallel string processing which could potentially run on a GPU: https://www.microsoft.com/en-us/research/publication/data-pa...

carapace on June 28, 2019 | parent | prev | next [–]

Well, it's APL, but uh...

https://github.com/Co-dfns/

AMA: Explaining my 750 line compiler+runtime designed to GPU self-host APL (youtube.com) https://news.ycombinator.com/item?id=13797797

danielecook on June 27, 2019 | prev | next [–]

Here is an alternative solution, although yours is a good one.

If you only need a single SNP or a group of SNPs within a region, you can use tabix[1] to index gzipped TSVs and query by genomic position. The position of SNPs can be obtained from a lookup table (2.5M is not very big even for R) or from an API if you were to say - query by rsid.

tabix also works over http (and s3) and can utilize RANGE queries to select a subset of a file...so you only wind up downloading a or reading a small portion once it is indexed and can do something like this:

tabix https://www.file.url.tsv chr1:1-1000

The command above would return variants on chromosome 1 between 1 and 1000.

The following variant browser works in this way: https://elegansvariation.org/data/browser/ - Theres no formal database (e.g. MySQL) running here, just tabix (actually bcftools which uses tabix) to select variants in a particular region, wrap them in JSON, and return to the client.

Setting this up on S3 requires configuring CORS... the igv browser also uses tabix indexes and provides guidance on how to set this up [2]

[1] https://www.htslib.org/doc/tabix.html [2] https://github.com/igvteam/igv.js/wiki/Data-Server-Requireme...

I created a similar solution to what you have done using this alternative approach, writing a wrapper in R that invoked bcftools under the hood. The dataset I was working with was a lot smaller (1.6M Snps x 252 individuals), but should work with larger genotype sets as well.

xiaodai on June 27, 2019 | prev | next [–]

I was onto a very similar idea recently! I used split to split a large CSV file and then used disk.frame (https://github.com/xiaodaigh/disk.frame), which is my package, to read in a large file. Not 25TB though!

markus_zhang on June 27, 2019 | prev | next [–]

I found this article particularly interesting as the author discusses a lot of (failed) methods.

Since I never dealt with big data before, I'm wondering what would you do in this situation?

> marcinzm on June 27, 2019 | parent | next [–]
>
> It seems he had a lot of issues due to Spark executors failing which seems a setting issue. My guess is that the executors were being killed by the system OOM killer. Spark's memory management is counter-intuitive. Spark spills intelligently to disk so executors don't need a lot of memory to process data if you're not doing interactive queries. However spark will use all the memory it's given and sometimes it will use more than that (might be the OS actually, not sure). So the trick is to give Spark's executors LESS memory (as a percentage of the node's memory) so there's a buffer in case Spark uses more memory than allocated.
>
> > nstrayer on June 27, 2019 | root | parent | next [–]
> >
> > Pretty much. I am sure if I truly understood the inner workings of spark I would have been able to get it to work. I didn't go into it too much in the article but I did tweak the executor memory a lot. Going as far as transcribing the aws article on tuning into an R script that generated a config exactly as they stated. Also when I tried GLUE with its supposedly no-configure setup I still got the same problems.
> >
> > > marcinzm on June 27, 2019 | root | parent | next [–]
> > >
> > > Interesting. Another reason I can think for it failing is lack of disc space on the nodes. Spark will spill data to disc if it doesn't fit into memory and your nodes may not have had enough disc space for 25TB of data.

emmanueloga_ on June 27, 2019 | prev | next [–]

Unix pipelines, AWK, gnu parallel, R, all great stuff.

If you have such an specific task, why not just write an "actual program" (as opposed to pipeline of scripts)? From the looks of it, it sounds like this problem could have been solved with, say, 50 lines of Java, C, Go, etc, etc. Maybe a bit more verbose but it would give you full control, you wouldn't need to lookup how to use command line parameters on S/O, and would probably give you a bit more performance.

> IndustrialJane on June 27, 2019 | parent | next [–]
>
> I have done that more than once. I often end up with a solution that works on the test set but which breaks after 10 TB just because <" "@example.com> is a valid email address according RFC-822 (Who the f* thought it was a good idea to allow spaces in email addresses?). Or some other exception that was not part of the test set, and that was not identified before starting.
>
> Dealing with exceptions is extremely error prone if these exceptions are not mapped beforehand. Thus it can be very costly.
>
> Similarly doing stuff in parallel is extremely error prone due to race conditions: What does not happen when running on your 1 GB test set, may very well happen when running on your 25 TB production data.
>
> > emmanueloga_ on June 27, 2019 | root | parent | next [–]
> >
> > I get your point but the same error handling problems can appear in scripts and pipelines, no?
> >
> > In a program I'd try/catch defensively "just in case", if missing one line out of 25TB is not a bit deal.
> >
> > For parallel processing I'd reach for the nearest standard library at hand on the language of choice.
> >
> > > IndustrialJane on June 28, 2019 | root | parent | next [–]
> > >
> > > > For parallel processing I'd reach for the nearest standard library at hand on the language of choice.
> > >
> > > That is a good example of what I mean: The nearest standard library is likely to either buffer output in memory or not buffer at all (in which case you can have the start of one line ending with another line). This means you cannot deal with output bigger than physical RAM. And your test set will often be so small that this problem will not show up.
> > >
> > > GNU Parallel buffers on disk. It checks whether the disk runs full during a run and exits with a failure if that happens. It also removes the temporary files immediately, so if GNU Parallel is killed, you do not have to clean up any mess left behind.
> > >
> > > You _could_ do all that yourself, but then we are not talking 50 lines of code. Parallelizing is *hard* to get right for all the corner cases - even with a standard library.
> > >
> > > And while you would not have to look up how to use command line parameters on S/O you _would_ be doing exactly the same for the standard libraries.
> > >
> > > Assuming you can get better performance is also not given: GNU Sort has built-in parallel sorting. So you clearly would not want to use a standard non-parallelized sort.
> > >
> > > Basically I see you have 2 choices: Built it yourself from libraries, or build it as a shell script from commands.
> > >
> > > You would have to spend time understanding how to use the libraries and the commands in both cases, and you are limited by whatever the library or the command can do in both cases.
> > >
> > > I agree that if you need tighter control than a shell script will give you, then you need to switch to another language.
> > >
> > > > emmanueloga_ on June 28, 2019 | root | parent | next [–]
> > > >
> > > > I agree with everything you said, as always, everything is a trade off. Good point about trickiness of memory management w/parallel processing! Would have to be extra careful to avoid hoarding RAM.

> flukus on June 28, 2019 | parent | prev | next [–]
>
> > If you have such an specific task, why not just write an "actual program" (as opposed to pipeline of scripts)?
>
> With the pipeline you get free parallelism and it's much easier to iterate over the individual steps, run step manually, check the output, add it to the script. You can also trivially break bits into a make file for improved parellelism and incrementalism. Performance wise these tools have had a lot of work put into them, even when they're not the most efficient tool on paper they'll often beat out the most naive versions in "real" languages.

epistasis on June 10, 2019 | prev | next [–]

One note, this type of data isn't traditionally considered "sequencing" data, as it's a much simpler form of point measurements. Yes, you get the base call at a place of common variation, but "sequencing" is generally reserved for those cases where one gets extended sequences of bases, rather that just data from here and there.

totalperspectiv on June 27, 2019 | prev | next [–]

Welcome to Bioinformatics! Excellent write up!

parhamn on June 27, 2019 | prev | next [–]

While I get the whole "you can do it on one node without all the complexity" thing, I do still wonder if map-reduce-synchronizer + coreutils is better than the behemoths that are the distributed ETL platforms right now. All the system would need to do is make a data file available on a node and capture stdout of the unix pipeline. I know gnu parallel does some of this.

fpbarthel on June 29, 2019 | prev | next [–]

Just wondering if this problem could have been solved by a properly indexed table? The article says: “Eight minutes and 4+ terabytes of data queried later I had my results“. 4+ TB seems way too much for 60k patients and sounds like an inefficient table scan was performed.

> fpbarthel on June 29, 2019 | parent | next [–]
>
> Also, wouldn’t partitioning only make sense if there is a sensible way to separate data that is more likely to be accessed vs data less likely to be accessed? Like is common with date data, since recent entries are often more relevant compared to old entries. For example, you could categorize SNPs by priority and eg. partition SNPs of high importance (frequently accessed) vs medium importance (sometimes accessed) vs low importance (rarely accessed).

marcinzm on June 27, 2019 | prev | next [–]

One correction to the article, snappy is not splittable however parquet files using snappy ARE splittable. Parquet compresses blocks of data within a file rather than compressing the file as a whole. Each block can then be read and decompressed independently.

sansnomme on June 27, 2019 | prev | next [–]

What's the best pipeline/workflow management tool for command line programs with a GUI? E.g. for resuming the process after it gets interrupted etc.

> c0l0 on June 27, 2019 | parent | next [–]
>
> I am not kidding: A graphical terminal emulator.
>
> Mastering the usual command line interface (terminal emulator, interactive shell, maybe a terminal multiplexer) is non-optional if you want to use CLI tools at or close to peak effectiveness.
>
> > sansnomme on June 27, 2019 | root | parent | next [–]
> >
> > I mean task resumption after interruption etc. Like airflow type of tools. Not quite unix task suspend options, this is about data pipelines. For Hadoop-style MapReduce, you can split the task into jobs which can be resumed and discarded etc. Shell scripting is not an elegant way to deal with this, a proper orchestrator tool is better.
> >
> > > cure on June 27, 2019 | root | parent | next [–]
> > >
> > > You could try the tool my group builds, Arvados (https://arvados.org). We use Common Workflow Language (CWL) as the workflow language. Arvados works great for very large computations and data management at petabyte scale. It really shines in a production environment where data provenance is key. Intelligent handling of failures (which are inevitable at scale) is a key part of the Arvados design.
>
> samuell on June 27, 2019 | parent | prev | next [–]
>
> Pipeline mgmt tool with a GUI, or tool for cli tools with a GUI? ;)
>
> One very widely used option is Galaxy https://galaxyproject.org
>
> You might also want to check out:
>
> - Arvados https://arvados.org
>
> - Chipster https://chipster.csc.fi
>
> - Knime https://www.knime.com
>
> They each have their strenghts and weaknesses.
>
> I'd probably be going with Galaxy as the default option, unless you find specific reasons to go with some of the others.
>
> dredmorbius on June 27, 2019 | parent | prev | next [–]
>
> Screen, if it's the connection session itself. Tmux if you're new-school.
>
> Task-splitting itself is inherently recoverable, as incomplete work units don't produce the final output you're looking for, and can. be retried, re-run, or re-entered into the task pool.
>
> A GUI quite frankly simply gets in the way.
>
> rodgerd on June 27, 2019 | parent | prev | next [–]
>
> CMake has a GUI (https://www.johnlamp.net/cmake-tutorial-3-gui-tool.html), and Make is pretty much perfect for your use case.

srean on June 27, 2019 | prev | next [–]

Can someone add "join" to the language/standard please. Its awkward to have split but not join.

dima55 on June 27, 2019 | parent | next [–]

There's a "join" tool in the GNU coreutils. "man join"

EForEndeavour on June 27, 2019 | root | parent | next [–]

Somehow, I hadn't known of the existence of the `join` utility until this moment. I really should devote some time to play around with it and paste, sort, awk, etc.

enriquto on June 27, 2019 | parent | prev | next [–]

what do you mean, exactly? Isn't this just concatenation?

dredmorbius on June 27, 2019 | parent | prev | next [–]

man (1) paste

```
paste - merge lines of files
```

jstrong on June 27, 2019 | prev | next [–]

(my) Lesson Learned: don't use spark.

scottlocklin on June 27, 2019 | prev [–]

You can solve many, perhaps most terascale problems on a standard computer with big enough hard drives using the old memory efficient tools like sed, awk, tr, od, cut, sort & etc. A9's recommendation engine used to be a pile of shell scripts on log files that ran on someone's desktop...

derefr on June 27, 2019 | parent | next [–]

I was writing a batchwise ETL tool (to break documents in some proprietary format down into rows to feed to Postgres's COPY command) and I achieved a remarkable level of IO parallelism by relying on Unix tooling to do my map-reducing for me.

1. I wrote a plain SQL *mapper* program, which spawns a worker-thread pool, where each worker opens its own "part" file for each SQL table, such that a document consumed by worker N gets records written to "tableA/partN.pgcopy".

2. And then, after the mapper is done, to do the reduce step, I just spawned a `sort -m -u -k n1` invocation to collate the row-files of each table together into a single `.sql` file. This not only efficiently merge-sorts the (presorted) "part" files into one file (without needing to sort the files themselves), but also blows away any rows with duplicate primary-keys [i.e. duplicate first columns in the row's TSV representation]—meaning I can restart the mapper program in the middle of a job (causing it to create a new set of "parts") and sort(1) will take care of cleaning up the result!

I honestly don't think anything could be done to make this system more optimal for its use-case. sort(1) goes *crazy* fast when it can mmap(2) files on disk.

(Also, I'm pretty sure that even the framework part of the mapper program—the "N persistent workers that each greedily consume a document-at-a-time from a shared pipe, as if they were accept(2)ing connections on a shared socket"—could be created with Unix tooling as well, though I'm not sure how. GNU parallel(1), maybe?)

Bonus: once you have SQL rowsets in TSV form like this, you can calculate a "differential" rowset (against a rowset you've already inserted) using `comm -23 $new $old`. No need for a staging table in your data warehouse; you can dedup your data-migrations at the source.

ramses0 on June 27, 2019 | root | parent | next [–]

Look into "bash-reduce", but it'd be great to have something like "bark" (bash-spark) which consumed documents at a time... and you're right, it might not even be that difficult.

banku_brougham on June 29, 2019 | root | parent | prev | next [–]

Could you share the code for these operations? I have some similar occasional use cases.

dredmorbius on June 27, 2019 | parent | prev | next [–]

In the waning months of G+ I fired off a few bulk archives (front page only), not the full community post set of the top 100k or so Communities (size and recency criteria) using a one-liner of awk, xargs, curl, and the Internet Archive's Save Page Now USL (SPN: https:web.archive.org/save/$URL). That took a bit over an hour.

On a mid-1990s iMac running Debian, and a rusty residential DSL connection.

R played a role in other community-related analysis and reporting.

wongarsu on June 27, 2019 | parent | prev | next [–]

For anything more complicated you can also get very far with simple python programs that read one line at a time and output some transformation of it (which might include turning one line into many to be piped into sort etc)

markus_zhang on June 27, 2019 | root | parent | next [–]

I think that the optimal way to do these kind of things is:

1) Assuming there is no joins/merges requirement, read in chunks and output GB dumps.

2) If joins/merges are required, use external merge sort.

Is this correct? Actually I'm wondering whether I could earn some bread and butter by focusing on the big data processing problems (e.g. sort/filter Terabytes+ dumps, do transformation for each line for Terabytes+ dumps, those kind of things) without actually knowing how to implement math algorithms (required for data science).

If so what kind of tools I need to master? I'm thinking about basic *nix tools like mentioned above, and also Python and maybe some compiled language for optimization (someone managed to speed up a Python external merge algorithm on 500GB file by

50% by implementing in Go), then maybe some easy algorithms (merge join, heap, etc.)

acomjean on June 27, 2019 | root | parent | next [–]

Learning Unix tools is pretty good place to start. There are a lot of commands that can do a lot of processing. It's been a while since I learned but the book "Unix power tools" from oreily is pretty good. It's old, but honestly these commands haven't changed much.

http://shop.oreilly.com/product/9780596003302.do

Python is slower compared to some of it's compiled cousins, but it's quick to write and a great skill to have when bash scripting can't handle some of the complexity or you need dB access. We use it sometimes to call c programs to do DNA sequence alignments and process the returns.

markus_zhang on June 27, 2019 | root | parent | next [–]

Thanks a lot! Time to fire up VirtualBox and learn some things.

Anon84 on June 27, 2019 | root | parent | prev | next [–]

Congratulations. You've just discovered the basics of MapReduce :)

adrianN on June 28, 2019 | root | parent | next [–]

Minus the enormous overhead that MapReduce brings.

devonkim on June 27, 2019 | parent | prev | next [–]

Furthermore, as computers get faster and cheaper in every dimension what makes economic sense to use "Big Data" tooling and efforts gets substantially larger with it. The limits of single nodes 15 years ago were pretty serious but most problems businesses have even in the so-called enterprise can currently easily fit on a workstation costing maybe $5k and be crunched through in a couple hours or maybe minutes - a lot easier to deal with than multiple Spark or Hana nodes. Operationalizing the analysis to more than a single group of users or problem is where things get more interesting but I've seen very, very few companies that have the business needs to necessitate all this stuff at scale - most business leaders still seem to treat analytics results in discrete blocks via monthly / weekly reports and seem quite content with reports and findings that take hours to run. Usually when some crunching takes days to run it's not because the processing itself takes a lot of CPU but because some ancient systems never intended to be used at that scale are the bottleneck or manual processes are still required so the critical path isn't being touched at all by investing more in modern tools.

I can support "misguided" Big Data projects from a political perspective if they help fund fixing the fundamental problems (similar to Agile consultants) that plague an organization, but most consultants are not going to do very well by suggesting going back and fixing something unrelated to their core value proposition itself. For example, if you hire a bunch of machine learning engineers and they all say "we need to spend months or even years cleaning up and tagging your completely unstructured data slop because nothing we have can work without clean data" that'll probably frustrate the people paying them $1MM+ / year each to get some results ASAP. The basics are missing by default and it's why the non-tech companies are falling further and further behind despite massive investments in technology - technology is not a silver bullet to crippling organizational and business problems (this is pretty much the TL;DR of 15+ years of "devops" for me at least).

0x5002 on June 27, 2019 | root | parent | next [–]

That is precisely what the projects I'm usually involved in do. A client might want "buzzword technology", but at the heart of it, what they really need are stable, scalable, and consolidated data pipelines to e.g. Hadoop or AWS that gives "Data Scientists" a baseline to work with (and anyone needing information, really - it was just called "Business Intelligence" a couple of years ago).

In the end it doesn't matter if you wind up with a multi-TB copy of some large database or a handful of small XML files - it's all in one place, it gets updated, there are usable ACL in place, and it can be accessed and worked with. That's the point where you think about running a Spark job or the above AWK magic.

awiesenhofer on June 27, 2019 | root | parent | prev | next [–]

> most business leaders still seem to treat analytics results in discrete blocks via monthly / weekly reports and seem quite content with reports and findings that take hours to run.

I would go further and even call long or at least not instant report generation a perceived feature. Similar to flight and hotel booking sites that show some kind of loading screen even if they could give instant search results, the duration of the generation itself seems to add trust to the reports.

edraferi on June 27, 2019 | root | parent | prev | next [–]

> The basics are missing by default

Absolutely. I really want to see advanced AI/ML tools developed to address THIS problem. Don't make me solve the data before I use ML, give me ML to fix my data!

That's hard though, because data chaos is unbounded and computers are still dumb. I think there's still tons of room for improvement though.

devonkim on June 27, 2019 | root | parent | next [–]

I watched a talk by someone in the intelligence community space nearly 8 years ago talking about the data dirt that most companies and spy agencies are combing through and the kind of abstract research that will be necessary to turn that into something consumable by all the stuff that private sector seems to be selling and hyping. So I think the old guard big data folks collecting yottabytes of crap across the world and trying to make sense of it are well aware and may actually get to it sometime soon. My unsubstantiated fear is that we can't attack the data quality problem with any form of scale because we need a massive revolution that won't be funded by any VC or that nobody will try to tackle because it's too hard / not sexy - government funding is super bad and brain drain is a serious problem. In academia, who the heck gets a doctorate for advancements in cleaning up arbitrary data to feed into ML models when pumping out some more model and hyperparameter incremental improvements will get you a better chance of getting your papers through or employment? I'm sure plenty of companies would love to pay decent money to clean up data with lower cost labor than to have their highly paid ML scientists clean it up, so I'm completely mystified what's going on that we're not seeing massive investments here across disciplines and sectors. Is it like the climate change political problem of computing?

dgacmu on June 27, 2019 | root | parent | next [–]

> In academia, who the heck gets a doctorate for advancements in cleaning up arbitrary data to feed into ML models

Well - Alex Ratner [stanford], for one: https://ajratner.github.io/

And several of Chris Re's other students have as well: https://cs.stanford.edu/~chrismre/

Trifacta is Joseph Hellerstein's [berkeley] startup for data wrangling: https://www.trifacta.com/

Sanjay Krishnan [berkeley]: http://sanjayk.io/

devonkim on June 27, 2019 | root | parent | next [–]

I was asking somewhat rhetorically but am glad to see that there's some serious efforts going into weak supervision. At the risk of goalpost moving, I am curious who besides those in the Bay Area at the cutting edge are working on this pervasive problem? My more substantive point is that given the massive data quality problem among the ML community I would expect these researchers to be superhero class but why aren't they?

dgacmu on June 27, 2019 | root | parent | next [–]

... they are?

There are a lot of people tackling bits and pieces of the problem. Tom Mitchell's NELL project was an early one, using the web in all its messy glory...http://rtw.ml.cmu.edu/rtw/

Lots of other folks here (CMU). Particularly if you add an active learning. Hard messy problem that crosses databases and ML.

lordleft on June 27, 2019 | parent | prev | next [–]

AWK is such an amazing tool

coliveira on June 27, 2019 | root | parent | next [–]

Awk is the most useful tool that people largely ignore in the UNIX tool chest. If you think of any script that has simple logic and involves transformations on input data, it could be more easily written in awk and integrated with the shell. After learning awk your UNIX abilities will increase exponentially.

mooreds on June 27, 2019 | root | parent | prev | next [–]

I found this article to be really great for outlining all the capabilities of awk:

https://developer.ibm.com/tutorials/l-awk1/

From 2001.

fian on June 28, 2019 | root | parent | next [–]

Seconded, and the follow up articles:

https://developer.ibm.com/tutorials/l-awk2/ https://developer.ibm.com/tutorials/l-awk3/

These were mentioned but not linked to in the previous form of the article/blog, had a quick look at the newer version you linked to and that may still be the case.

A lot of the Awk info I had found prior to stumbling on these articles was focused on command line one-liners. So the sections on defining Awk scripts as files and multiline records were a great help to me.

Something1234 on June 27, 2019 | parent | prev | next [–]

What is A9?

hayksaakian on June 27, 2019 | root | parent | next [–]

It's an Amazon subsidiary

https://en.wikipedia.org/wiki/A9.com

leksak on June 28, 2019 | parent | prev [–]

What's A9?