Making Massive Computational Experiments Painless

Hatef Monajemi[*], David L. Donoho[†] and Victoria Stodden[‡]
[*]Department of Statistics, Stanford University
Stanford, California 94305
Email: monajemi@stanford.edu
[†]Department of Statistics, Stanford University
Stanford, California 94305
Email: donoho@stanford.edu
[‡]School of Information Sciences, University of Illinois at Urbana-Champaign
Champaign, IL 61801
Email: vcs@illinois.edu

Abstract—**The increasing availability of access to large-scale computing clusters - for example via the cloud - is changing the way scientific research can be conducted, enabling experiments of a scale and scope that would have been inconceivable several years ago. An ambitious data scientist today can carry out projects involving several million CPU hours. In the near future, we anticipate a typical Ph.D. in computational science may be expected or even required to offer findings based on at least 1 million CPU hours of computations.**

The massive scale of these soon-to-be-upon-us computational experiments demands that we change how we organize our experimental practices. Traditionally, and still the dominant paradigm today, the end-to-end process of experiment design and execution involves a significant amount of manual intervention and situational tweaking, cutting and pasting, and the use of disparate disconnected tools, much of which is undocumented and easily lost. This makes it difficult to detect and understand possible failure points in the computational workflow, making it virtually impossible to correct, let alone simply rerun the experiment. This is an amazing state of affairs, considering the ubiquity of error in scientific computation and in research generally. Following such unstructured and undocumented research practices limits the ability of the researcher to exploit cluster and cloud-based paradigms, as each increase in scale under the dominant paradigm is likely to lead to ever more errors and misunderstandings.

A better paradigm will integrate the design of large experiments seamlessly with job management, output harvesting, data analysis, reporting, and publication of code and data. In particular such a paradigm would submerge the details of all the processing, harvesting, and management while exposing transparently the description of the discovery process itself, including details such as the parameter space exploration. Reproducing any job would be a push-button affair, and creating a new experiment from a previous one might involve only changes of a line or two of code followed again by push-button execution and reporting. Even though such experiments would be operating at a much greater scale than today, under such a paradigm they would be easier to conduct, obtain a lower error rate, and offer a much greater opportunity for 'outsiders' to understand the results.

In this article, we discuss the challenges of massive computational experimentation and present a taxonomy of some of the desiderata which such paradigms should offer. We then present ClusterJob (CJ), an efficient computing environment that we and other researchers have used to conduct and share million-CPU-hour experiments in a painless and reproducible way.

Index Terms—**Reproducible Research, High-throughput Computing, Big Data**

Traditionally, science has involved two paths to knowledge: deduction - formalized through proofs in the mathematical sciences - and induction - measurement and hypothesis driven experimentation in the physical and biological sciences. Today, a third avenue is emerging based on massive scientific computation. Throughout research in the sciences and technology we observe a significant use of large-scale computation: numerical wind tunnels, global circulation models, Kaggle machine learning challenges, numerical oil field reservoir simulations, and so on.

Whole disciplines are emerging where the important research questions can only be studied by massive computational experimentation. In field after field, questions like "which algorithm performs better" or "which configuration is optimal" or "what would be the trade-off between goals A and B" are the main questions of the moment, and can now *in principle* only be addressed by large-scale computational experimentation. Mathematical analysis cannot provide a complete answer because it has the power to study only very idealized (simplified) situations and very idealized (simplified) interventions. Physical experimentation is inapplicable in many cases because the questions concern systems described in silico or in historical databases rather than laboratory systems which can be manipulated physically or chemically [1].

[1]In some cases true experimentation is possible as in A/B testing of web interfaces for example, however large scale computation is emerging as an indispensable tool for a large variety of research questions.

Even if physical experiments may be plausible in some cases, they may be economically infeasible.

Successful examples of computation as a fundamental method of scientific discovery abound. In [1] massive computations solved a 30 year old puzzle in the design of a particular protein, in [2], several million CFD simulations are conducted to find optimal design strategies for oil field development, and in [3] and [4], we used more than a million CPU hours to discover and document fundamentally more practical sensing methods in the area of Compressed Sensing.

Prediction challenges offer a rapidly expanding arena of scientific activity that is also cementing the primacy of massive computation. Well-known examples include the protein structure prediction challenge [5] and the Netflix challenge [6]. In a prediction challenge, a specific processing task is formalized and some training data are made public while some test data are sequestered. Teams develop predictive models based on the public data and submit their models to be scored robotically on the sequestered test data. The scores are presented on a leaderboard and the winner is the leader at the conclusion of the challenge. Challenges have served as the engine of progress over the last two decades of research in language processing, object detection and biometrics. Examples are Multiple Biometric Grand Challenge (MBGC) [7] and Large Scale Visual Recognition Challenge (ILSVRC) [8]. A long series of such challenges improved for example the accuracy of speech-to-text systems culminating in successful consumer speech recognition apps, such as Siri, Echo and the Android's SpeechRecognizer.

Much of the current buzz about deep learning is caused by the success of certain labs[2] using deep learning tools to winning numerous challenges; this has invested deep learning with great authority and charisma, and made it a popular method for various prediction tasks [9], [10].

Note that the traditional tools of scientific epistemology – mathematical analysis and physical experiment – are effectively irrelevant to winning prediction challenges; but the ability to run massive numbers of computational experiments is integral to success.[3]

The emergence of massive computation as a fundamental strategy for the generation of new knowledge is driven by the explosion of computational resources globally, as evidenced by open-source software, ubiquitous internet connectivity, massively distributed databases, and cloud computing. Some globally significant organizations (AWS, Google, Microsoft) manage clusters with *millions* of CPUs, which an individual can purchase.

Despite the amazing capacity for brute work that such facilities offer, and the economic feasibility for many researchers, the perceived complexity of using such facilities is a barrier preventing many scientists from conducting ambitious computational research at scales which are now possible.

Many of today's active researchers have grown up in the era of laptop- and desktop- computers running quantitative programming environments like Matlab and R. Such scientists subscribe to a paradigm of *interactive computing*, where small snippets of code are pasted into the console listener and output plots and printouts are studied, and maybe a bit of text or graphics is cut and pasted from the output window into a digital notebook before the next interactive task is formulated. Later, using a word processor, the researcher turns author and will copy-paste items from the notebook into the technical report describing the experiment's results.

Interactive computing is intuitive, intimate, familiar and has been relentlessly promoted both in academic training, in academic texts, and by makers of quantitative programming environments. It also matches the habits and mindset many people have developed from interacting with their smartphones over e-mail and web browsing. Many working scientists imagine that today's interactive computing paradigm is the best of all possible worlds and couldn't imagine wanting to operate otherwise.

This dominance of this 'interactive computing' paradigm is the biggest barrier we face in explaining the impending transition to large-scale computer experiments. In our opinion, the unspoken idea that it is natural to run snippets of code interactively and to cut and paste results into word processing files, while pervasive, is also *problematic scientific ideation*. No-one who thinks this way could possibly emerge a winner of one of today's important prediction challenges. Nor would someone who thinks this way be able to complete a large-scale 1 million CPU hour computing project of the kind now becoming standard for Ph.D's in the computational fields; they would no doubt be unable to keep track of the massive volume of work required, to fix the inevitable errors that crop up in anyone's large-scale projects, resulting in either an unfinished project, or a set of incorrect results.

It's utterly crucial to move the research community toward the following three new expectations about the role of computing in modern research:

1) The most central contribution a researcher can make is to dream up new *computational experiments* - a success metric, a baseline system, and a set of computations exploring variations of the baseline model, for example varying underlying parameters and/or models and improving the given metric.

[2]An example is the Geoffrey Hinton's lab in University of Toronto who won the Kaggle's Merck Molecular Activity Challenge.

[3]Moreover, winning one single challenge might conceivably be a fluke; but when a long series of seriously-designed challenges succeeds in transforming a classifier error rate from (say) 50% to (say) 1%, it's hard to claim that challenges are not engines of scientific progress.

2) A researcher can make her/his reputation by carrying out such an experiment exhaustively and at scale with impeccable technique.
3) The "royal road" to scientific reputation goes through assembling and/or building tools enabling impeccable technique and outstanding productivity in carrying out such massive computational experiments.

In some fields, especially in machine-learning prediction challenges, we believe these three expectations are already endemic. Because of what we are about to say next, we believe that these three expectations will develop over time in field after field.

We are confident that a new paradigm is emerging that will make massive computationally intensive experiments painless; it will also make it possible to really understand what took place in such large experiments, to believe in the definitive nature of results of such experiments, and begin to depend on them as reliable science in the way we today depend on mathematical proofs. The new paradigm - once it crystallizes and spreads widely - will drastically improve the quality and productivity of computational science. We don't claim to fully formulate or exhaustively discuss this paradigm in this short paper, but we do hope to explain our own efforts pointing in this direction, and to contrast them with other efforts known to us.

In our telling, a computational experiment involves:

1) *Precise specification* which includes defining a performance metric and a range or grid of different systems to be compared.
2) *Distribution and management* of all the jobs implicitly required in 1).
3) *Harvesting* of all the data generated by all the jobs in 2).
4) *Analysis* of the data produced by 3).
5) *Reporting* and dissemination of results.

To truly exploit the power of massive computations, it becomes crucial to automate all these components; squeezing out all the friction, complexity, and sources of ambiguity and misunderstanding. Automation of computational experimentation has long been considered for discovery [11].

In our opinion, the sought-after paradigm will take the form of an *experiment-definition system* (EDS), where the conduct of the experiment follows inexorably and automatically from the mere definition of the experiment itself. In detail such an EDS would satisfy the following desiderata:

- **Legibility**: The EDS allows the specification of experiments and their end goal in a human-readable way which stands apart from details about how the experiments take place.
- **Simplicity**: The EDS should be easy to use, setting off large amounts of computational work essentially as easily as pushing a single button.
- **Productivity**: Roughly speaking, all the tedium that has been historically involved in running large-scale computing experiments, such as the baby sitting involved in managing job execution and data harvesting, will be obviated. Researchers will instead spend their time on designing interesting experiments and reporting on their outcomes.
- **Durability:** Experimental processes and outcomes will be preserved in such a way that can be easily retrieved at a later time.
- **Transparency**: It can easily be understood post facto where each component data value came from, what the actual parameters of every computation were, etc. The system is in some sense self describing, including about what processes have run, how their execution went, and what data were accessed.
- **Reproducibility**: All the required tasks happen in a reproducible way. Computations can be rerun, and even variations of computations can be forked where needed. The code and data underlying a computation can be effortlessly exposed to outsiders whenever the researcher wishes to do so; and because of the transparency just discussed, outsiders can understand and build upon computational results.
- **Scalability**: The EDS should accommodate massive scaling-up of experiments, moving from one cluster to another, or to the cloud with ease.

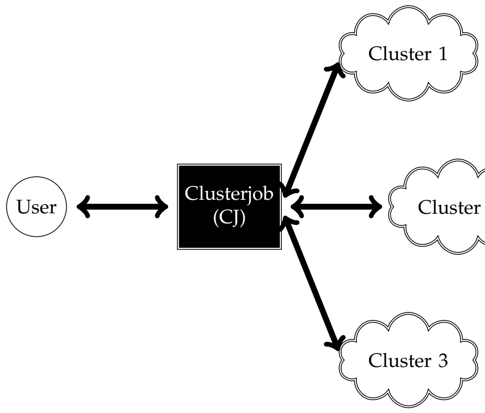

Fig. 1. Cartoon of ClusterJob

To facilitate massive computational experimentation for our research, we have developed ClusterJob (CJ) [12], an automated computing environment that makes inroads into implementing each of these features.

I. ClusterJob

ClusterJob (CJ) is an open-source software system that allows researchers to conduct massive computational experiments in a painless and reproducible way. CJ is used in our research to run more than *50 million reproducible* *computations* on Stanford campus cluster. One can think of CJ as a computational agent who is responsible for scaling up and farming out all the computations across compute clusters, tracking their progress, harvesting and analyzing the produced data, and disseminating reproducible results (see Figure 1).

CJ is built on the idea of producing reproducible computational packages with distinct *P*ackage *Id*entifiers (PIDs). In the simplest case, a researcher writes a main script, say in MATLAB, that implements a research question. The script and possible dependencies are handed to CJ, who is responsible for creating a reproducible computational package with a specific PID (a SHA-1 code), allocating resources to execute the script, and finally updating the researcher on the status and results of the run. Through their PIDs, the CJ-confirmed packages can be published to a website, and shared with and reproduced by other independent researchers when needed.

CJ is easy-to-use and requires very little training. In addition, the users do not have to learn a new programming language on top of what they frequently use (e.g., MATLAB, R, Python).

We can demonstrate CJ with an example. Consider running an explanatory experiments involving many variation of two parameters to calculate certain quantity of interest, which in our example is the probability of successful recovery by a convex optimization algorithm. We first define the experiment in a simple and decipherable MATLAB script, say `test.m`:

```
% test.m
% This test code calculates the
% probability of successful
% reconstruction in compressed sensing.
% Author: Hatef Monajemi Nov 1 2016

file = 'results.txt';
delta   = 0.1:.1:.9;
epsilon = 0.02:0.02:0.98;
for i = 1:length(delta)
for j = 1:length(epsilon)
  pr = computeProb(delta, epsilon);
  fid = fopen(file,'at');
  fprintf(fid, '%3.2f,%3.2f,%3.2f\n', ...
                delta,epsilon,pr);
  fclose(fid)
end
end
```

This script computes probabilities at 441 points on a `(delta, epsilon)` grid of size 9×49, and outputs a text file (named `results.txt`). To run this experiment in parallel on 441 cores of a remote cluster (named `corn`) is as easy as typing the following CJ command in your laptop's console:

```
$ cj parrun test.m corn -dep bin -m "Test PT"
```

where `bin` is a directory that includes required dependencies such as `computeProb.m` code.

Upon execution of this command, CJ submits 441 independent runs in a *reproducible* way to the `corn` cluster and provides the user with a PID, which is a 40 digits long hexadecimal number. Having the PID, one can track the progress of the runs, harvest the data, and get other information about the experiments at any time using various commands provided by CJ's command line interface. A typical short information log for an experiment may look like:

```
pid 8ab7a5aafa1b8232cc3da05a7814bed1d21dd0aa
date: 2016-Oct-08  11:47:37  (GMT -07:00:00)
user: monajemi
agent: 2DCA5476-8197-11E6-B8C8-3A835C8A0BAC
account: monajemi@corn.stanford.edu
script: test.m
initial_flag: parrun

          Test PT
```

An example of harvesting produced data is:

```
$ cj reduce results.txt 8ab7a5aa
```

which amalgamates all the data from individual runs into a single `results.txt` file. The results can also be stored in other accepted MATLAB output formats such as `.mat` files.

In essence, the users do not feel much of a change; they write their codes the way they normally would for running on their local machine, yet CJ allows them to run and manage massive amounts of computations on powerful compute clusters in an effortless and reproducible way.

The use of ClusterJob leads to more productive research and helps researchers be more efficient. Here is a list of what we consider to be the most important benefits of using CJ:

- *Facilitating submission of jobs to clusters*: CJ removes the need for the researcher to be familiar with clusters. All that is needed is to know CJ commands, which are straightforward.
- *Work organization and easy access to old computations*: Many of us have experienced the trouble of finding a code from an old project. When a computation is done through CJ, the relative path and other information regarding various computations are stored by CJ and can easily be queried through their PID.
- *Automatic parallelization*: CJ can speed up computations by automatically distributing the task among many cores of a computational cluster as long as the tasks are independent (i.e., embarrassingly parallel).
- *Automatic reproducibility*: CJ automatically generates computational packages in a reproducible way; hence making computations reproducible comes at no-cost.
- *Easy sharing*: The packages can be shared with others via their PIDs in a simple and straight-forward way.

- *Elimination of researcher's degrees of freedom*: Clusterjob restricts researchers from changing the results of computations. This prevents problems like p-value hacking and data dredging.

To learn more about Clusterjob, the reader is referred to http://clusterjob.org or the GitHub page https://github.com/monajemi/clusterjob.

II. Other available systems

Several other systems exist today that help facilitate computational experimentation in different ways. Some that we are aware of include[4]:

- *CodaLab*[13]: A system for reproducible research that uses Docker containers [14] to keep the full provenance of an experiment in the form of immutable *bundles*, which can later be included in *worksheets* for final reporting. It also allows inclusion of data from old bundles to new runs in a straight-forward manner. By default, CodaLab runs the experiments on Windows Azure workers.
- *SDM* [15]: A collection of tools developed by Stanford VISTA lab for harvesting and analysis of neuroimaging data.
- *Image Harvest (IH)* [16]: A high-throughput image processing system for the outputs of high-throughput plant phenotyping platforms.
- *VisTrails*: A scientific workflow and provenance management system that captures and makes available the computational steps in a research workflow. [17].
- *Kurator*: A system that automates data curation within the Kepler scientific workflow system [18].
- *Torch*: A scientific computing framework for machine learning algorithms on GPUs that uses LuaJIT scripting language. The project is available online via http://torch.ch
- *Sumatra*: A toolkit for automatically capturing the details of computational experiments for reproducibility [19].
- *Pegasus:* A workflow management system that manages experiments deployed over distributed data and computer resources. It is specifically designed to operate at scale and deploy, for example, many millions of computational steps in a chain of experiments [20]. Pegasus was employed for example in the recent LIGO Gravitational Waves discovery [21].

The focus in many of the systems mentioned above is to facilitate "reproducibility" rather than to ease "scalability". They provide computing environments that ensure capturing the full record of a computational experiment but do not give any clue to researchers on how they can scale up their computational endeavors in a painless and systematic way so as to assure knowledge. Those that do provide tools for scalability (e.g., IH and Torch) are either specialized or pay little attention to the reproducibility aspect of experimentation. We believe that both reproducibility and scalability are equally important for massive computational experiments and indeed they are highly intertwined: many issues associated with reproducibility will be automatically resolved en route to developing pain-free systems for massive computations.

The contours of the new computing paradigm for doing massive computational experiments are clear but the details can vary. We have presented our own effort to build this new paradigm but many other implementations are plausible that satisfy the desiderata presented in this article, and so we anticipate that there will be many candidates before the scientific community comes to an agreement.

[4]This list is not meant to be exhaustive.

Acknowledgments

The ClusterJob project was supported by National Science Foundation grants DMS-0906812 (American Reinvestment and Recovery Act), DMS-1418362 ('Big-Data' Asymptotics: Theory and Large-Scale Experiments), and DMS-1407813 (Estimation and Testing in Low Rank Multivariate Models).

References

[1] P. Huang, K. Feldmeier, F. Parmeggiani, D. A. Fernandez-Velasco, B. Höcker, and D. Baker, "De novo design of a four-fold symmetric TIM-barrel protein with atomic-level accuracy," vol. 12, pp. 29–34, 2015.

[2] M. G. Shirangi and L. J. Durlofsky, "Closed-loop field development under uncertainty by use of optimization with sample validation," *SPE Journal*, vol. 20, no. 05, pp. 908 – 922, 2015.

[3] H. Monajemi, S. Jafarpour, M. Gavish, S. C. . Collaboration, and D. L. Donoho, "Deterministic matrices matching the compressed sensing phase transitions of Gaussian random matrices," *PNAS*, vol. 110, no. 4, pp. 1181–1186, 2013. [Online]. Available: http://www.pnas.org/content/110/4/1181.abstract

[4] H. Monajemi, "Phase transitions in deterministic compressed sensing, with application to magnetic resonance spectroscopy," Ph.D. dissertation, Stanford University, 2015. [Online]. Available: https://purl.stanford.edu/gf738wr7593

[5] "Critical Assessment of protein Structure Prediction." [Online]. Available: http://predictioncenter.org

[6] "Netflix prize." [Online]. Available: http://www.netflixprize.com

[7] P. J. Phillips, P. J. Flynn, J. R. Beveridge, W. T. Scruggs, A. J. O'Toole, D. Bolme, K. W. Bowyer, B. A. Draper, G. H. Givens, Y. M. Lui, H. Sahibzada, J. A. Scallan, and S. Weimer, *Overview of the Multiple Biometrics Grand Challenge*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 705–714. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01793-3_72

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, pp. 436–444.

[10] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces."

[11] D. Waltz and B. G. Buchanan, "Automating science," *Science*, vol. 324, no. 5923, pp. 43–44, 2009. [Online]. Available: http://science.sciencemag.org/content/324/5923/43

[12] H. Monajemi and D. L. Donoho, "Clusterjob: An automated system for painless and reproducible massive computational experiments," 2015. [Online]. Available: https://github.com/monajemi/clusterjob

[13] "Codalab, accelerating reproducible computational research." [Online]. Available: http://codalab.org
[14] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2600241
[15] B. A. Wandell, A. Rokem, L. M. Perry, G. Schaefer, and R. F. Dougherty, "Data management to support reproducible research," 2015, arXiv:1502.06900.
[16] A. C. Knecht, M. T. Campbell, A. Caprez, D. R. Swanson, and H. Walia, "Image harvest: an open-source platform for high-throughput plant image processing and analysis," 2016. [Online]. Available: http://jxb.oxfordjournals.org/content/67/11/3587.full
[17] L. Bavoil, S. Callahan, P. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo, *VisTrails: Enabling interactive multiple-view visualizations*, 2005, p. 18.
[18] L. Dou, G. Cao, P. Morris, R. Morris, B. Ludscher, J. Macklin, and J. Hanken, "Kurator: A kepler package for data curation workflows," *Procedia Computer Science*, vol. 9, pp. 1614 – 1619, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050912002980
[19] A. P. Davison, M. Mattioni, D. Samarkanov, and B. Telenczuk, *Sumatra: a toolkit for reproducible research*, 2013, p. 19.
[20] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus: a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015, funding Acknowledgements: NSF ACI SDCI 0722019, NSF ACI SI2-SSI 1148515 and NSF OCI-1053575. [Online]. Available: http://pegasus.isi.edu/publications/2014/2014-fgcs-deelman.pdf
[21] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, and S. Koranda, "Griphyn and ligo, building a virtual data grid for gravitational wave scientists," in *High Performance Distributed Computing (HPDC)*, 2002. [Online]. Available: http://pegasus.isi.edu/publications/ewa/hpdc11.pdf