Making Technological Timelines: Anticipatory Repair and Testing in High Performance Scientific Computing

Sims, Benjamin Hayden

To be published in: Continent

DOI to publisher's version:

Permalink to record: http://permalink.lanl.gov/object/view?what=info:lanl-repo/lareport/LA-UR-16-21095

Making Technological Timelines: Anticipatory Repair and Testing in High Performance Scientific Computing

Benjamin Sims

Think of some examples of repair in everyday life. Maybe you had a car accident and took your car to the body shop. Maybe the head came off your child's doll and you had to glue it back on. Maybe the handle of your shovel cracked and you wrapped the cracked area with duct tape to hold it together. These are examples of what could be called *reactive repair*, where an unexpected accident initiates a sequence of action and decision-making that ends in repair. In these cases, most of the thinking and planning surrounding repair takes place after a breakdown has been identified. This type of repair is often taken to be distinct from deliberate design, as it occurs within the context of technology that is already in operation, often has an improvisational character, and may be performed by end users or technicians rather than credentialed experts. But does repair always have to be reactive? And if not, what does this tell us about the distinction between design and repair, and their respective roles in shaping technological change? The short answer is that repair, like design, can play a dynamic and forward-looking role in shaping technological trajectories – not only stabilizing existing systems, but anticipating change and generating new technological futures.

Repair practices in large technological systems and infrastructures reveal different levels of planning for breakdown and repair. At the most basic level, a general need for ongoing maintenance and repair of a system can be foreseen and provided for – what is often called *routine maintenance*.[1] One step further along the planning spectrum is *preventive maintenance*, which involves predicting specific kinds of breakdowns in advance and trying to prevent them from happening.

A final level of planning, which I call *anticipatory repair*, deploys a much more diverse and technically sophisticated array of resources to predict breakdowns and develop approaches for preventing or responding to them. This is a distinctly modern regime of repair, premised on science and engineering methods for forecasting failure. This type of repair aligns with a broader contemporary trend toward anticipation in the governance of science and technology, one that emphasizes the virtues of optimization and preparedness.[2] The instruments of anticipatory repair include modeling and simulation, user studies, trials, testing, and other methods that aim to instigate breakdowns in a controlled setting before a technology is deployed for general use, thus affording an opportunity to fix problems before they affect end users.

Anticipatory repair is an important part of the work of scientific High Performance Computing (HPC) code developers, who design software to carry out complex computing tasks, such as large physics simulations. (HPC is now the preferred term for what has also been called supercomputing.) Specifically, developers of long-lived codes anticipate the architectural features of future HPC machines, and start adapting codes to run on them before they are built.

Although HPC systems are now made from off-the-shelf components, they are nonetheless far from a standardized commodity. Pushing to the next level of computing performance almost always involves introducing innovative or unusual architectural features. Since HPC codes are highly optimized to the hardware they run on, getting a code to run effectively on new hardware can be challenging and require significant investment of resources, which can impact scientific productivity. This is an ongoing problem because new hardware is typically deployed every 5-7 years, while complex scientific codes can be actively developed and used for decades.

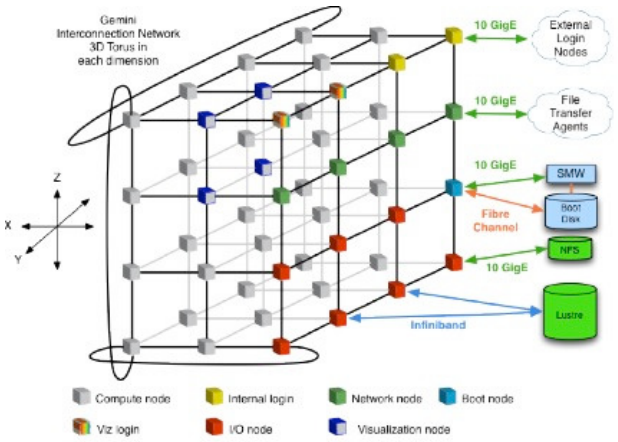

Figure 1. Typical modern HPC system design. Source: "Cielo Computational Environment Usage Model With Mappings to ACE Requirements for the General Availability User Environment Capabilities," Los Alamos National Laboratory Report number LA-UR-12-24015.

An example of how HPC code developers prospectively adapt codes to new hardware is the work that went into deciding what type of new machine to develop for basic science research at Los Alamos National Laboratory. The choice was between two options:

1. A machine with specialized many-core Intel Xeon Phi processors (similar to the Trinity system shown in Figure 2), which would be more powerful but might require significant modifications to many existing codes.
2. A less powerful system with more conventional processors that would require minimal code changes.

Figure 2. Trinity computer construction at Los Alamos National Laboratory. Source: http://www.lanl.gov/newsroom/picture-of-the-week/pic-week-14.php.

To help make this decision, managers called on the expertise of the Scientific Computing Support Team (SCST), a group of expert code developers who provide advice to scientists on programming challenges. They asked the team to select a few representative computing projects, analyze their codes, and work with the code developers to see how difficult it would be to make the codes run efficiently on Xeon Phi processors. I spent a year working with this team during this process, and conducted interviews with team members and code developers about their efforts.

The SCST's job was, first, to test how the unmodified codes ran on Xeon Phi processors, and, second, to try various approaches to optimizing the

codes to run efficiently on those processors. A number of works in science and technology studies have focused on the role of tests, trials, and demonstrations in the development and deployment of technological systems.[3] Trevor Pinch notes that testing often involves projection from a test situation (say in a laboratory) to some kind of "real world" use scenario.[4] This projection is usually justified by asserting that the test situation is similar enough to the use scenario to justify inference between the two, even though they may differ in scale, timing of events, presence of measuring devices, etc. Projection usually has a temporal aspect, in that it may connect the test to events that occur before, concurrent with, or after it.

There were some obvious challenges to projection between the SCST's testing and the actual performance of codes on a full-scale HPC system based on Xeon Phi processors. In particular, the team only had access to a few of the current version of the Xeon Phi processor on a local cluster, while an actual system would likely have hundreds of processors, and would use a version of the processor that hadn't been released yet.

The SCST's strategy for establishing the relevance of their small-scale testing was to work closely with code developers to come up with test problems that would be small enough to run in a reasonable length of time on a few processors, yet still be representative of code performance on a full-size HPC machine. This typically involved developing a smaller or simpler version of the physical system the code would normally simulate, but which had similar computational properties. For example, for a molecular dynamics code:

"With [the code developer's] help, and a lot of back and forth, we came up with a 128-atom molecular simulation. It took maybe a dozen back-and-forths before we came to a problem that was large enough to fit in the memory footprint yet give some scalability."

To support projection from performance on the current version of the processor to performance on the unreleased version, SCST members focused on relevant similarities in the hardware and programming work involved:

"The problems we face in getting code to perform on [the current version] will be the same problems we face with [the unreleased version]. Even though the hardware is different, the fact that we have so many cores is the same challenge."

Having established a reasonable test problem, the next step was to set the code up to run on the Xeon Phi processor. This was a complex and often frustrating challenge, and the initial result was almost always that the code ran drastically slower on the Xeon Phi processor. The extent of this degradation was the first indicator of how the code would perform on the processor, and established a worst-case scenario for the type of breakdown that could be anticipated when the codes were run on a new machine.

This is where testing starts to converge with breakdown and repair: the first stage of testing was explicitly oriented toward provoking a technological breakdown and measuring its impact. By simulating a potential future breakdown in the present, the team created a displacement in time that gave decision-makers an opportunity to include that breakdown in their assessment of potential future systems.

The next step was to test possibilities for repair. To do this, the SCST experimented with different ways of configuring the code to see which resulted in optimal performance. Most of this performance tuning was done by experimenting with different settings for compilers and parallelization tools, which affect how computations are distributed across cores and processors. The end result, in most cases, turned out to be that the code could be made to run as fast or faster on the Xeon Phi processor as on a more conventional processor, without having to make major changes to the code itself. This result could then be projected forward in time to suggest that any potential breakdowns that resulted from running current codes on a new Xeon Phi system could be repaired without too much trouble.

This was reassuring from a decision-making standpoint, but equally importantly, the testing actually enacted the steps that would need to go into any future repair, establishing relevant skills and relationships. This testing and repair effort, then, served to knit together present and future states of the HPC ecosystem – not just assessing the outcome of a potential technological trajectory,

but actually contributing to its realization.

Conclusion

A number of contributions to this volume draw attention to the role of repair and maintenance in shaping what Ribes and Finholt have called the "long now" of infrastructures,[5] using examples as diverse as the lifecycle of mobile phones (Houston), the temporal frames of long-term space missions (Cohn), and the "technological residues" left behind by a failed software project (Fiore-Gartland). Anticipatory repair describes a particular future-oriented mode of repair that deploys diverse predictive resources to align present interests with imagined technological futures. In so doing, it also contributes to the realization of those futures. In this mode of repair, the "broken world thinking" described by Jackson[6] is leveraged in service of a planned and controlled approach to technological innovation and progress.

Anticipatory repair is an example of the kind of anticipatory practice Adams, Murphy, and Clarke identify as central to the current scientific and political moment. Anticipation work is temporally dynamic, enabling a "tacking back and forth between futures, pasts, and presents, framing templates for producing the future."[7] The HPC example presented here suggests that this "tacking back and forth" in time may play an essential role in enabling stabilization and control over complex infrastructural systems over long time periods. In this context, testing and anticipatory repair emerge alongside design as powerful tools for establishing technological trajectories and managing uncertainty about technological futures. Reactive repair complements anticipatory repair by cleaning up after the breakdowns and catastrophes it inevitably fails to prevent. Both forms of repair can contribute to the stabilization of technological systems in time and the generation of new technological possibilities. Repair is about more than restoring lost order – it can also be a powerful tool for anticipating, projecting, and shaping future worlds.

REFERENCES

[1] For example, the work performed by technicians in Christopher Henke's "The Mechanics of Workplace Order: Toward a Sociology of Repair," *Berkeley Journal of Sociology* 44 (2000): 55–81.

[2] Vincanne Adams, Michelle Murphy, and Adele E. Clarke, "Anticipation: Technoscience, Life, Affect, Temporality," *Subjectivity* 28 (2009): 246–65; Guston, David H., "Understanding 'Anticipatory Governance.'" *Social Studies of Science* 44, 218–242 (2014).

[3] E.g. Donald MacKenzie, "From Kwajalein to Armageddon? Testing and the Social Construction of Missile Accuracy," In *The Uses of Experiment: Studies in the Natural Sciences*, ed. David Gooding, Trevor Pinch, and Simon Schaffer (Cambridge, UK: Cambridge University Press, 1989), 409–435; Benjamin Sims, "Concrete Practices: Testing in an Earthquake-Engineering Laboratory," *Social Studies of Science* 29, no. 4 (1999): 483–518; Christopher R. Henke, "Making a Place for Science: The Field Trial," *Social Studies of Science* 30, no. 4 (2000): 483–511.

[4] Trevor J. Pinch, "'Testing - One, Two, Three ... Testing!': Toward a Sociology of Testing," *Science, Technology, and Human Values* 18, no. 1 (1993): 25–41.

[5] David Ribes and Thomas A. Finholt. "The Long Now of Technology Infrastructure: Articulating Tensions in Development," *Journal of the Association for Information Systems* 10 (May 2009): 375–98.

[6] Steven J. Jackson, "Rethinking Repair," in *Media Technologies: Essays on Communication, Materiality, and Society*, ed. Tarleton Gillespie, Pablo Boczkowski, and Kirsten Foot (Cambridge, MA: MIT Press, 2014), 221-239.

[7] Adams, Murphy, and Clarke, "Anticipation: Technoscience, Life, Affect, Temporality," 246.