Harnessing Complexity in High Performance Computing Ecosystems: A Complex Adaptive Systems Framework

Nan-Chen Chen[1], Lavanya Ramakrishnan[2], Sarah S. Poon[2], Cecilia Aragon[1]
[1] Department of Human Centered Design & Engineering, University of Washington, Seattle, WA, USA
[2] Lawrence Berkeley National Laboratory, Berkeley, CA, USA
nanchen@uw.edu, LRamakrishnan@lbl.gov, sspoon@lbl.gov, aragon@uw.edu

Abstract

The use of high performance computing (HPC) has been generating influential scientific breakthroughs since the twentieth century. Yet there have been few studies of the complex socio-technical systems formed by these supercomputers and the humans who operate and use them. In this paper, we describe the first complex adaptive systems (CAS) analysis of the dynamics of HPC ecosystems. We conducted an 18-month ethnographic study that included scientific collaborations that use an HPC research center and examined the processes in HPC socio-technical systems via CAS theory to devise organizational designs and strategies that take advantage of system complexity. We uncovered several significant mismatches in the variation and adaptation processes within subsystems and conclude with three potential design directions for management and organization of HPC socio-technical ecosystems.

1. Introduction

Computational science has been producing significant scientific breakthroughs since the twentieth century. Numerous fields [1-3] rely on advanced computing technologies to understand and solve complex problems. High performance computing (HPC), or supercomputers, play an important role in scientific discovery. Incorporating thousands of nodes linked by powerful networks to support inter-node communication, HPC systems are capable of running extremely large-scale simulations and analyses in parallel. Along with traditional computational science, data intensive discovery, recently termed the "fourth paradigm" of scientific discovery [4], is projected to continue to gain influence as the amount of data in the world expands exponentially. The necessity of processing vast and expanding amounts of scientific data with HPC systems has also been predicted to expand. However, barriers to HPC efficiency go beyond computational benchmarks and involve human interaction, human efficiency, and human-scale time [5]. Therefore, studies of HPC operators and users interacting with their supercomputers are critically needed. Such studies, however, need to take into account the complexity of the underlying socio-technical systems comprising multiple large machines and the interactions of hundreds or thousands of humans; a reductionist view focused on a single factor such as efficiency or usability may miss important details.

HPC machines form a part of a large "socio-technical ecosystem" where people (e.g. scientists, engineers, staff) collaboratively interact with the machines, and structures within the hardware, software, and the human organization impact how people, processes, and machines influence each other. We define the cross-disciplinary term "socio-technical ecosystem" based on terminology from multiple fields, originating from "socio-technical system" [6, 7] and "technological system" [8]. Here we view an ecosystem as comprising the complex interactions between people, machines, and their environments. The term "ecosystem" emphasizes the organic nature of system components, with a focus on how they constantly change and evolve. The analogy of a natural ecosystem has been used in multiple fields, e.g. business ecosystems and software ecosystems, to describe how member organisms interact and co-evolve in their environment [9].

The HPC socio-technical ecosystem (hereafter, HPC ecosystem) involves complex social and technical interactions among its participants. For example, scientific users are focused on their research output. On the other hand, HPC staff are responsible for the effective and efficient utilization, maintenance and evolution of the supercomputing systems that serve diverse scientific communities, with sometimes conflicting requirements. Computer engineers work closely with both these groups to develop software that serves the needs of the users, while also focusing on performance gains. Since the HPC ecosystem is an open ecosystem where people can enter and leave the ecosystem, the decision-making process and hence, co-evolution of these parts of the ecosystems, are complex and dependent on a number of internal and external factors. As all components in the HPC ecosystem are highly interdependent, it is not easy to parse and explain the individual interactions and phenomena.

Complex adaptive system (CAS) theory, an approach that focuses on relationships, patterns, and processes within a dynamic system, provides a means to unpack some of the intricacy and interdependency within the HPC ecosystem.

In this paper, we present the results from an 18-month ethnographic study that included scientific collaborations that used HPC resources in the United States, and develop a framework to examine the HPC ecosystem from a CAS point of view. We highlight sets of agents, patterns and interactions we observed in our field study. Drawing upon the results, we present potential design directions for the management and organization of HPC socio-technical ecosystems.

2. Background

2.1. Complex adaptive systems

Complex adaptive system (CAS) theory takes an evolutionary perspective to the study of systems [10-12]. Due to the interdependencies and constant changes in this type of system, a CAS may be difficult to comprehend and predict. According to Axelrod and Cohen [13], a CAS "consists of parts which interact in ways that heavily influence the probabilities of later events." The goal of CAS theory is to harness complexity, not to eliminate it—to provide a framework to consider the design of organizations and strategies that takes advantage of this complexity without needing to fully understand and control each detail in the entire system.

While CAS is not a single theory, there are a few characteristics that are fundamental: First, CAS focuses on complex systems involving highly interdependent components, or subsystems, with dynamic connections, in contrast to simple systems that consist purely of the sums of their components. Changes within such a system may also be non-linear and cannot be easily decomposed. Second, CAS emphasizes the evolutionary processes of system elements. Third, a CAS will contain self-organizing and emergent behavior, involving no direct or central control of its processes, but the appearance of collectively emergent order and patterns within the ecosystem.

We define the elements and processes of CAS as follows, largely based on Axelrod and Cohen [13]: A CAS consists of multiple subsystems or components (e.g., [14, 15]). Each subsystem contains agents, or processes, that create or interact with artifacts. These agents may be grouped into types possessing shared properties.

Evolutionary processes of variation, interaction, and selection are constantly occurring in the subsystems. Variation can be either exploitative or explorative. Exploitation refers to a variation that requires minimal process changes to achieve certain goals, such as adopting a well-known solution in a community. Exploration, on the other hand, is a variation which can be very different from the original state, and usually has no existing example as reference.

Variations may arise from interaction between agents (e.g., one agent copies a strategy from another agent and modifies it), and variations may also create new possibilities of interaction. The selection process comes with a set of success criteria to measure and ultimately change the frequency of types. According to Axelrod and Cohen [13], when a selection process leads "to improvement according to some measure of success," it is called adaptation.

Agents in one subsystem may interact with agents in another subsystem, or may also interact with artifacts created from still another subsystem. Therefore, a CAS heavily relies on and is impacted by the co-evolution of subsystems: Each subsystem evolves not only on its own, but also adapts to changes in other subsystems. In other words, the variations, interactions, and selection processes of one subsystem may be influenced by another system's processes.

2.1.1. Literature of complex adaptive systems. CAS theory has been applied successfully across widely divergent domains, such as healthcare [16, 17], nursing [5, 31], ecology [18-20], supply networks [10, 21], languages [22, 23], markets [24, 25], organization management [11, 26, 27], and software development [28]. CAS theory has been shown effective across multiple domains in describing system behavior when study targets are more than fixed mechanistic and predictable systems. Namely, agents in the system have autonomy and are self-organized. Patterns in the system emerge and are not the result of central controls. Moreover, these systems evolve to adapt to constant changes in their environments, or adapt to change in other agents.

For example, in the healthcare domain, Rouse pointed out that hierarchical decompositions of healthcare systems, i.e., describing them as linear and hierarchical compositions of parts, is ineffective [29]. One key reason is that healthcare systems possess no single authority or central control. Each stakeholder behaves according to their own potential interests and risks.

Like healthcare systems, HPC ecosystems cannot be completely controlled and centrally determined. The design of the machines themselves involve complex trade-offs, and no single authority ends up in charge of everything about the supercomputer. When a facility purchases an HPC system, the procurement process itself involves an intricate set of social and financial

interactions. HPC machines are designed by vendors based on a set of facility requirements. The vendors then issue proposals, which are evaluated by a committee within a lengthy procurement process involving multiple trade-offs, regulations, and considerations. The needs of the users are but one factor in the final design, purchase, and deployment. When the supercomputer is eventually deployed, the conflicting and ever-changing needs of its users add to the complexity of its operation. Clearly, HPC ecosystems fall into the complex systems category and cannot be modeled by a simple linear system.

Other research has focused on the co-evolution of agents and/or subsystems: Kim and Kaplan combine CAS and actor-network theory [30] to study university timetables and demonstrate the co-evolution of different subsystems. Briscoe finds that the language and language acquisition devices (i.e., brains) co-evolve [23]. Rammel et al. discuss the occurrence of co-evolutionary processes in the subsystems of natural resource networks, which include the resource base, social institutions, and individual agents [31]. Cherry examines economic sectors and finds that policymaking systems are involved in "a coevolutionary dance with other complex adaptive systems in society, including business and economic systems [32]." In this paper, we also focus on co-evolutionary processes among research, engineering, and facilities subsystems. Furthermore, we discuss the challenges of co-evolution, which occurs between the gaps of these subsystems. CAS theory has been widely applied across various domains, including organizational IT, although it has received less attention in the HPC field. Kaplan and Seebeck [12] adopted CAS to study 35 years of IT systems in a university, and drew parallels with computer-supported cooperative work design as a type of complex adaptive systems design. They then constructed a taxonomy of CAS terms as applied to IT.

To our knowledge, our work constitutes the first in-depth ethnographic study at an HPC research center that utilizes the CAS model to deliver insights into the complex human-machine collaboration that characterizes HPC ecosystems.

2.2. Research site, data collection and analysis

Our ethnographic study was conducted from September 2014 to March 2016 in the United States and included scientific collaborations that used supercomputers available at the National Energy Research Scientific Computing Center (NERSC). We interviewed 24 people from both the scientific collaborations and NERSC. The interviewees included 9 scientists, 8 engineers, and 7 HPC staff members. 22 interviewees were male and two were female.

All interviews were transcribed, cleaned, and coded by the first author. The coding process consisted of multiple steps. The first author read through the transcripts in Word, cleaned them up based on the audio recordings, extracted quotes considered informative and left comments to highlight or summarize significant paragraphs. Second, key CAS terms from Kaplan & Seeback's taxonomy [12], which were based on Axelrod and Cohen's [13], were used to construct the basic codebook. Then all the quotes and comments were extracted from Word files to an Excel spreadsheet, and an existing code from the basic codebook was applied to each quote or a new code was created. All quotes ended up with zero to four codes. Next, we identified the three subsystems in the HPC ecosystem. Thus, we arranged quotes and listed key CAS elements of each subsystem and put emphasis on co-evolution. Finally, we organized the quotes into themes regarding the challenges and gaps between co-evolutionary processes of the three subsystems.

2.3. NERSC machines

NERSC currently operates two major HPC systems: Edison and Cori. NERSC serves about 6,000 users and hundreds of projects. Every few years, NERSC starts a new procurement process for purchasing the next generation HPC system. NERSC generates a list of intended features, and vendors who design and sell HPC systems will submit proposals for NERSC to consider. After a proposal has been chosen, it will take a few years to construct, deliver, and deploy the machine and for NERSC staff and users to prepare for the new system to come online.

2.4. Roles and workflows in the HPC ecosystem

In this section, we provide a brief overview of the roles and workflows in the HPC ecosystem we studied. There are three primary roles: scientists who do research, engineers who develop software and help scientists with code development, and HPC staff members who maintain HPC systems and support machine-related issues.

A scientist's key expertise lies in a particular domain that they were trained in (e.g., material science, climate science, physics), and they usually work in groups under research projects, that are funded through grants. Engineers or computer scientists are highly skilled in areas of high performance computing, software development or other areas of computer science and usually work on multiple research projects across scientific domains to support the computing needs of scientists Their jobs can range from developing an independent software package to helping scientists debug software ("codes"). Some

HPC staff members maintain machines, and some are responsible for supporting users, such as helping users to set up jobs and install software package dependencies. Large science projects often consist of scientists, engineers and sometimes HPC staff members who build tools and technologies towards a common goal. Not all scientists in a project may use HPC systems.

Allocations of computational time and/or storage resources at NERSC are awarded to scientists on a project basis. Scientists wrap their codes into jobs, specify their requested resource amount, and submit their jobs to NERSC systems. Once the jobs are submitted, they are placed in a queue (multiple queues have different priorities and CPU hour charges) to wait for execution. As each job arrives at the front of the queue, the system allocates the resources the user requested and executes the job.

Each scientific domain is accustomed to their own set of software packages. Some of these packages are generic, such as NumPy and SciPy in Python, whereas some simulation packages or visualization tools are domain-specific. In order to utilize HPC systems, scientists may need to parallelize their codes, where parallelization might include language-dependent complexities. For example, for C and Fortran users, libraries like OpenMP [33] support parallelization well. However, for Python, since the native Python interpreter is not thread-safe (may produce consistency errors in shared data structures during parallel operation), users must use MPI4Py [34] or OpenMP together with Cython (an optimizing static complier to enable C-extension in Python) [35], which can bring extra complexity into workflows of scientists who use Python.

3. High performance computing ecosystem as a complex adaptive system

In this section, we identify three essential subsystems of the HPC ecosystem: the research, engineering, and facilities subsystems, each of which has its own CAS elements and separate processes, but can also interact with and rely on other subsystems. As a result, the subsystems not only evolve individually, but also co-evolve and adapt to the changes of other subsystems. We highlight important elements of the subsystems and their interactions in Figure 1.

3.1. Research subsystem

The research subsystem includes agents in the HPC ecosystem whose primary focus is to use supercomputers for scientific research. Most of these agents are research scientists, postdocs, and graduate

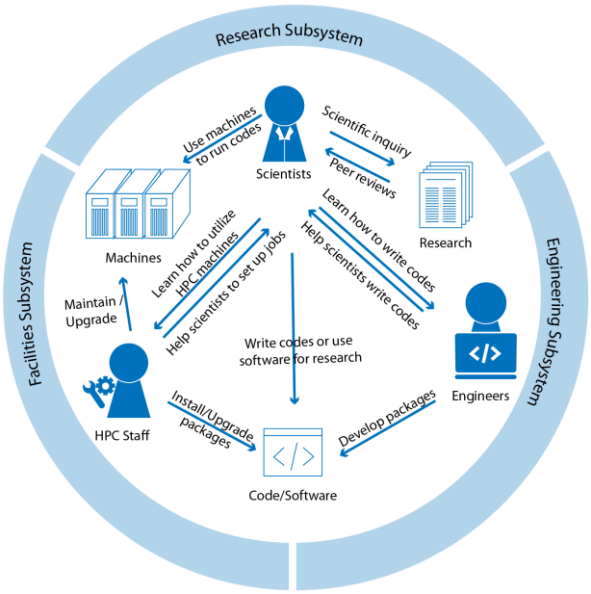

Figure 1. Summary diagram of the subsystems. The research subsystem consists of scientists and the artifacts they create or use. The engineering subsystem comprises engineers and the codes and packages they develop. The facilities subsystem includes the HPC staff and the machines. All subsystems interact with each other and may lead to variations. For example, a scientist can learn how to write more efficient codes from engineers.

students (all hereafter referred to as *scientists*). In addition to agents, the research subsystem contains various artifacts such as the research itself, source code and software packages, HPC machines, as well as the scientists' local machines (e.g., laptops and Linux workstations).

The research subsystem constantly cycles through the evolutionary processes of variation, interaction, and selection. The nature of scientific inquiry drives variation towards innovative research contributions. This includes both exploitation and exploration. For example, incremental research can be considered a case of exploitation of prior work, whereas introduction of a new technique (e.g., parallel programming) to a research area for the first time is a case of exploration. These cases can lead to the creation of new types of scientists (e.g., scientists who know parallel programming evolve to be called computational scientists and often have primary appointments in computational divisions), strategies (e.g., ways to deal with data), as well as artifacts (e.g., new software packages or hardware).

Taking another view, variation in this subsystem occurs through interaction within the subsystem as well as between subsystems. For instance, scientists in the same research project may form new strategies to divide work, or a discussion with other scientists may prompt a new research idea. Scientists may also improve their research through peer reviews (i.e., feedback from other scientists). In addition, a scientist may learn new programming techniques from engineers, or learn how to better utilize HPC machines from the HPC staff. For example, scientists may pick up computer science skills critical to their subsystem, as when we witnessed a scientist describing how to make his Python code accessible to people who use R, and his longtime engineer collaborator commented, "You [the scientist] are speaking my language right now."

Although numerous cases of variation may occur, the selection process can limit or even eliminate variations based on success criteria. In the research subsystem, the most important success criterion is scientific discovery, which itself can be measured by various criteria, such as the quantity and impact of research publications. Toward this end, scientists may prioritize research directions based on the possibility of influential outcomes.

Another common success criterion relies on resource utilization. Namely, scientists usually have limited resources, whether human (e.g., working hours, collaborator availability) or computational (e.g., CPU hours, memory, storage space). Thus, variation that exceeds a scientist's resources is unlikely to be sustained or even to appear.

3.2. Engineering subsystem

In the engineering subsystem, engineers, whose primary job is to develop software to support scientists, are the key agents. As with the research subsystem, the engineering subsystem also contains many artifacts, but among the most essential are source code and software packages.

Variation in the engineering subsystem is largely driven by scientists' needs and facility changes. For example, engineers who develop a software package may receive new feature requests from scientists, or engineers may have to help scientists parallelize their code. In such cases, variations are usually exploitative, adding changes incrementally following software development practices. However, in some cases, variations can also be explorative. For instance, engineers may develop a new software package from scratch. They may even completely refactor a software package to increase the performance and maintainability of the package. Furthermore, when HPC facilities change, engineers must modify their software or upgrade its dependencies to ensure it continues to function.

Through interaction with scientists, engineers learn their goals and habits in order to provide better support. They may also develop different methods of interaction to better meet scientist needs. One engineer we interviewed said his group interacted with scientists on a weekly basis to ensure that new software features would meet scientists' immediate needs. Another engineer pointed out that pair programming (i.e., sitting with a scientist to debug code) was routine practice. In addition to interacting with scientists, engineers also interact with other engineers, or work with HPC staff to ensure packages they develop are compatible with the newest machines.

There are multiple success criteria in this subsystem. How well engineers support scientists to enable them to harness scientific discoveries, which may not be explicitly defined, is one key success criterion. Often, engineers need to support more than one group of scientists, and they may be responsible for both developing software packages and facilitating scientists' code-writing processes. Therefore, this success criterion may be approximated through the quantity of issues engineers help a specific group of scientists to resolve. As a result, engineers may prioritize feature requests from one group over another. A second success criterion of the engineering subsystem is the quality of the engineering work, as defined by recommended engineering practices, for example code modularization. Hence, engineers may pursue variations that completely change a software package's structure but offer no new features.

Although software quality is important, in the HPC ecosystem the improvement of scientific quality has greater weight. One engineer pointed out that refactoring code to increase the engineering quality without adding new features does not usually count as a contribution of engineering work. Thus, the evolutionary direction of engineering favors better support of science.

3.3. Facilities subsystem

The facilities subsystem is centered on the HPC machines themselves. The key agents in this subsystem include HPC staff, such as people who interact with supercomputer users, those who maintain HPC software and hardware, those who interact with HPC vendors (i.e. procurement), and those who analyze machine utilization and define policies (for simplicity, we refer to all these sub-categories as *HPC staff*).

The variation process in the facilities subsystem relies not only on the advance of HPC technology, but

also on the needs of scientists in the research subsystem. For example, HPC staff may install a new software package due to requests from scientists. However, due to the extremely high cost of equipment purchase and operation, the success criteria of the facilities subsystem must be based on more than the satisfaction of user needs. Among others, system utilization, security, cost, and energy consumption are also important success criteria. Benchmarks are often used to measure the computational performance of an HPC system. NERSC evaluates many aspects such as computational benchmarks and application performance, cost, and power consumption. HPC staff must also balance how much they expect users to change in order to use the machine efficiently versus increased performance gains from hardware upgrades.

> *There are trade-offs between doubling the memory or I/O bandwidth versus having users modify their codes. It's about understanding the cost-benefit of your actions as well as just the pure cost, too, which again goes back to analyzing the application to understand what it is that they really need. Can you push and budget in roughly the right way amongst the different components in the machine? [HPC staff member 3]*

During our field study, a new HPC system, Cori, was introduced. Cori possesses a few new features that differ significantly from its past two generations. One critical difference is that Cori's compute nodes are split into two partitions: data and HPC. The data partition aims to serve people who need high data throughput (i.e., heavy I/O), whereas the HPC partition consists of the traditional compute machines. NERSC staff had to find a way to serve its diverse users and balance budget and energy. However, the goal of reducing power consumption led to each CPU core containing less memory and instead supporting threading. To take advantage of the new design, users were expected to increase parallelism in their code. Thus, evolution in the facilities subsystem favored user support as well as innovation in hardware technology within cost constraints.

4. Challenges of co-evolution of subsystems

As briefly described in the previous section, the HPC ecosystem's three subsystems influence each other's directions of evolution, or more accurately, co-evolve. Nevertheless, co-evolution is not always a smooth process and conflicts can arise to hinder collectively emergent orders. Adaptation in co-evolution requires variation, and selection processes in subsystems all yield improvements. Nevertheless, in many cases, we found that when one subsystem evolves, it may be difficult for other subsystems to adapt. Highlighting such challenges are like pinpointing *reverse salient* [36, 37] in technological systems where reverse salient refers to a slowly developed component in the system that prevents the whole system from achieving its goal. This analytical approach surfaces the limits of the current system. In this section, we layout several key challenges of co-evolution and obstacles to adaptation to highlight current issues in the HPC ecosystem

4.1. One subsystem must remain in an older state

Adapting to changes of other subsystems requires variations in a subsystem, but sometimes one subsystem must remain in its current state and thus cannot follow the changes of other subsystems. For example, when the facilities subsystem deploys a new generation HPC machine that includes fundamental differences from previous generations, it requires the research and engineering subsystems to prepare source code and software packages that are compatible with the new machine. Similarly, changes in the codes can also come from within the subsystem – e.g., bug fixes. A group of scientists may continue to use an older version of their simulation models due to a variety of reasons. It may be due to compatibility with other subsystems that are outside of the HPC subsystem, or it may be to ensure fairness in comparisons.

> *What I would be worried about in terms of different model versions is what the model developers do if they bring in a different … if they somehow change, which might be including fixing a bug, the algorithm for figuring <an intermediate variable> out and if they changed out at a bit. That can do some rather dramatic things <to the model results>. The value difference may seem to be small, but when go over the historical time period, that matters a lot. [Scientist 2]*

Additionally, scientists write papers for submission to journals and conferences, but the review process can be lengthy, so much so that the engineering subsystem may have evolved (e.g., a software package they use may have a newer version) before they receive reviews. When they receive comments from reviewers asking for more analysis, they must run their code under the same environment again:

> *It happens a lot when they have some papers submitted for review, and then the review comes three or four months later. They want to be able to run the exact same script at that exact same time. [Engineer 6]*

As a result, even though new versions of software packages constitute a preferred variation based on the

engineering subsystem success criteria, the research subsystem may not agree with the variations, let alone change to adapt to them. Thus, there is a need to evolve yet maintain strong provenance records that also capture the connections between the subsystems.

4.2. Subsystems have mismatched evolution directions

In some cases, two subsystems may evolve in mismatched directions, forcing people to use workarounds to connect them. For instance, in the research subsystem, the programming language Python is increasingly used for scientific data analysis. Packages such as NumPy and SciPy provide powerful utilization functions for scientific data analysis. However, Python's modularization design does not fit well with multicore systems like NERSC machines, causing issues in the facilities subsystem. For example, each Python process running on the HPC machines reads its dependent packages. For a job using 40,000 cores, the dependencies must be loaded 40,000 times, amounting to excessive I/O overhead.

Another blocking factor was that the compute nodes, for the sake of flexibility, were designed to require users to give them all dependent packages; hence code with no dynamic importing was preferable. However, Python is designed to dynamically import packages.

To handle this mismatch, people used various workarounds, such as wrapping all dependencies into a .tar file and submitting the "tarball" with the job. One scientist wrote a tool to cache package locations each job needed on individual nodes and include this information along with the job. However, the scientist told us that when Cori came online, his workaround did not function with the new job queuing system on Cori. This involved significant time fixing the workaround.

Thus, subsystems do not always co-evolve in aligned directions. Oftentimes, workarounds are created to compensate for mismatches, but these can then be vulnerable to changes in either subsystem.

4.3. Limited time and resources block adaptation

Even if one subsystem were to follow the evolution of other subsystems, limited time and resources can block them from creating variations for adaptation. In the case of the simulation models, for example, the scientists ran the older version of the models for over two years, generating more than 3.1 PB of output data from more than 1,000 simulation runs. If scientists were to use the latest version of the model with the same number of simulations, they would need to run the models for another two years.

On the other hand, the cost of enabling the older version to run on Cori also exceeded the amount of time and resources available in the engineering subsystem. The simulation models were developed much earlier in older generation machines with different architectures, and the code was not expected to run with so many instances at the same time. Therefore, it was not written in a way that would easily fit multicore architectures. Furthermore, the models required specific versions of software dependencies which were not available on Cori or Edison.

One of the engineers tackled the issue of the models only to find there were already too many layers of previous fixes in the code. Since the models had existed for many years, various people worked on the software, sometimes adding code to ensure the models fit new needs or previous environment changes. A change in one part would break other workarounds. He referred to this difficulty as "technical debt," explaining that when people resort to hacks to make technology work without sufficient planning, these hacks become debts later on, making it increasingly difficult to make any modifications at all:

> *The first thing I try to do will be completely buried in technical debt. I can't change this because this and this and this were workarounds that were designed to work around debts and if I change that, they all become bad, they all stop working and there's just layers and layers of this stuff. [Engineer 2]*

This is an example of where a previous evolution of one subsystem (research) occurred independently of other subsystems but caused challenges for future co-evolution. Such cases are not rare in the HPC ecosystem. For example, when a new storage architecture was introduced at NERSC, a scientist mentioned that it would not be possible for them to change their software to utilize it because the code was written in the '80s or '90s, and it was unlikely that the code could be updated.

> *Because a lot of these codes that were written, they started themselves in the '80s or '90s or something. There are a couple of ones that started in the more modern era, but most of them are fossils. … Maybe some of those fossil codes will be able to update themselves for this type of architecture, but I think it is unlikely to happen. [Scientist 5]*

Since rewrites or changes to code are not recognized in the reward system, it can be too costly in terms of time and resources to make changes to the code to adapt to the new facilities subsystem.

4.4. Conflicts between success criteria of subsystems

Obstacles to adaptation may come from conflicts between the success criteria of different subsystems. For instance, the research subsystem values research contributions (i.e., publications and scientific results). As a result, scientists are less likely to spend time on efforts like enhancing the quality of their code. Often, scientists want code that runs and prefer not to spend a lot of time tuning performance. If it takes a long time to receive results, they manually submit jobs to the HPC systems and switch to other tasks (e.g., writing papers). An engineer pointed out that people do not spend the time to automate their workflows until manually setting up jobs and waiting becomes too difficult.

However, to the facility subsystem, utilization of the HPC machines is an important success criterion, which can be challenging whenever a new supercomputer is deployed. For example, utilizing Cori required better-parallelized code. Such constraints may block scientists from producing research. This tension could be reduced by improving the usability of existing software tools and providing tools to help scientists update their codes more efficiently.

Another conflict happens when the engineering and facilities subsystems introduce a new version of software packages or upgrade the operating systems (OS) of the HPC systems. As mentioned earlier, scientists may find it difficult to switch to the updated version. Nevertheless, from the perspective of HPC staff, keeping software and system OS up-to-date reduces bugs and improves the security of the systems, important success criteria for the facilities subsystem.

In the engineering subsystem, engineers may need to refactor a package to follow better software engineering practices, or simply to support new architectures, both of which are part of their success criteria and necessary but may not be a consideration for the scientists. For instance, one scientist told us that a module in a visualization package he used should not be rewritten because it had been tested for 20 years and they trusted the quality of the software:

> *I've been arguing about this with the guys in the software development group. You don't want to rewrite this stuff <the module>. This is tested. It's 20 something years old now and bulletproof. It's good software. [Scientist 4]*

5. Discussion: Potential design directions for HPC ecosystem management and organization

Although complex adaptive systems cannot be fully controlled and designed, CAS theory provides a concrete framework that can provide guiding principles to promote policies, strategies, and interactions to shape evolutionary processes [13]. In this section, we outline three potential design directions for the HPC ecosystem that we hypothesize will encourage positive co-evolution and adaptation between subsystems for emergent order. Further research is needed to test these directions.

5.1. Defining success criteria for adaptation between subsystems

Conflicts between success criteria among different subsystems are a key issue blocking adaptation and coordination across subsystems. Therefore, we hypothesize that it is critical to define success criteria that take into consideration all subsystems. For instance, currently the research subsystem does not value time spent on parallelizing codes or improving software, yet it is an important task to better utilize HPC machines and ultimately enhance scientific output. Data and software are increasingly vital to scientific discoveries, the code often containing significant intellectual content including highly specialized scientific knowledge. As such, the intellectual content of software needs to be recognized and valued. One way to do this is to stop defining success metrics for each subsystem separately and consider the entire ecosystem as a whole. Similarly, HPC facilities are largely judged by performance benchmarks. However, in a previous study, Chen et al. pointed out the importance of *collective time* in HPC design [5]. They argued that HPC designers should not only consider machine time, but also human time required to set up and use HPC machines. Thus, if ease of parallelization and usability were to be explicitly valued across all subsystems, it may encourage scientists to better parallelize their codes, leading to more efficient software, less technical debt, and more human time for scientific insight. Further research into metrics that take into account the interaction between subsystems will be needed.

5.2. Managing mismatches and workarounds

There will inevitably be conflict between subsystem success criteria. If mismatches appear and people need to create workarounds to connect gaps and mitigate related issues, it is important to clearly identify those mismatches and visibly manage workarounds.

Besides their vulnerability to changes in associated software, workarounds also have the potential to evolve into long-term solutions. For example, the

workaround one scientist created to wrap dependencies into a .tar file to accompany jobs became a package that is now available to his group. Increasing the visibility of this type of process may be helpful to other groups as well. Therefore, we hypothesize that identifying mismatches and managing workarounds in the HPC ecosystem may not only help prevent severe breakdowns, but also help increase chances of adaptation. For example, one way to manage workarounds might be to enable a community repository that allows other subsystems to contribute patches, scripts that can then be reviewed and approved for more general use.

5.3. Supporting cross-subsystem communication and increasing interactions

Support for cross-subsystem communication may lead to significant amelioration of subsystems' conflicting goals and assumptions. For example, scientists may have very good reasons why they don't want to upgrade to the latest software version, while HPC staff may have conflicting but equally valid reasons to upgrade. Lowering the barrier to interaction between agents in both subsystems by providing a means for lightweight, short-timeframe communication could provide significant mutual benefit by enabling more frequent interaction and negotiation. Today, communication between the subsystems occurs through structured mechanisms (e.g., occasional requirements workshops), semi-structured (e.g., help desk) and ad-hoc (e.g., through previously established relationships). These mechanisms either operate over a long timeframe or contain sufficient friction to impede optimal interaction. The HPC facility subsystem should consider increasing communication of other significant events (e.g., major system upgrades) and solicit regular, frequent input from other subsystems; this could substantially improve the management of mismatches and workarounds.

We hypothesize that developing affordances to lower the social barriers that may currently impede interaction and negotiation could facilitate communication between subsystems. Previous work has demonstrated that technological affordances can be created to achieve this goal [38]. For example, a communication interface that allows visualization of both human and machine effectiveness could be utilized to enable negotiation. This could also increase people's situational awareness of their actions and their effects on other groups.

The end result could lead to better quality workarounds and more negotiation around mismatches, with the potential for greater machine efficiency, increased human satisfaction, and overall improved effectiveness of the entire HPC ecosystem.

6. Conclusion

This paper presented the first use of CAS to explore the HPC ecosystem via an in-depth ethnographic study. The CAS framework enabled us to surface mismatches and breakdowns that exist in the current variation and adaptation processes within subsystems. Based on these insights, we presented three potential design directions for HPC ecosystems which may provide important guidelines to participants and stakeholders. Future work should focus on testing these hypotheses and developing metrics that take into account interactions between subsystems, design to mitigate mismatches via improved affordances for lightweight and frequent communication, and the reevaluation of cross-system success criteria. The goal is to help reduce barriers to variation and enable seamless adoption of new directions in HPC environments, and ultimately lead to the acceleration of scientific discovery across all domains utilizing high performance computing.

7. References

[1] *Exploring the universe with supercomputing*. Available: https://sciencenode.org/feature/exploring-universe-supercomputing.php

[2] J. R. Collins, R. M. Stephens, B. Gold, B. Long, M. Dean, and S. K. Burt, "An exhaustive DNA micro-satellite map of the human genome using high performance computing," *Genomics*, vol. 82, pp. 10-19, 2003.

[3] R. Nemani, P. Votava, A. Michaelis, F. Melton, and C. Milesi, "Collaborative supercomputing for global change science," *Eos, Transactions American Geophysical Union*, vol. 92, pp. 109-110, 2011.

[4] T. Hey, S. Tansley, and K. M. Tolle, *The fourth paradigm: data-intensive scientific discovery* vol. 1: Microsoft research Redmond, WA, 2009.

[5] N.-C. Chen, S. Poon, L. Ramakrishnan, and C. R. Aragon, "Considering Time in Designing Large-Scale Systems for Scientific Computing," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, 2016, pp. 1535-1547.

[6] C. P. Lee, P. Dourish, and G. Mark, "The human infrastructure of cyberinfrastructure," in *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, 2006, pp. 483-492.

[7] E. F.E., T. E.L., C. C.W., and V. M.. "Socio-technical systems," *Management Science Models and Techniques*, vol. 2, pp. 83-97, 1960.

[8] T. P. Hughes, "The evolution of large technological systems," *The social construction of technological systems:*

New directions in the sociology and history of technology, vol. 82, 1987.

[9] M. Seppänen, S. Hyrynsalmi, K. Manikas, and A. Suominen, "Yet another ecosystem literature review: 10+ 1 research communities," in *2017 IEEE European Technology and Engineering Management Summit* , 2017, pp. 1-8.

[10] T. Y. Choi, K. J. Dooley, and M. Rungtusanatham, "Supply networks and complex adaptive systems: control versus emergence," *Journal of operations management*, vol. 19, pp. 351-366, 2001 2001.

[11] K. J. Dooley, "A Complex Adaptive Systems Model of Organization Change," *Nonlinear Dynamics, Psychology, and Life Sciences*, vol. 1, pp. 69-97, 1997 1997.

[12] S. Kaplan and L. Seebeck, "Harnessing complexity in CSCW," in *ECSCW 2001*, 2001, pp. 359-378.

[13] R. Axelrod and M. D. Cohen, *Harnessing complexity: Organizational implications of a scientific frontier*: Basic Books, 2000.

[14] M. K. Richard and M. K. Simon, "Interpreting socio-technical co-evolution: Applying complex adaptive systems to IS engagement," *Information Technology & People*, vol. 19, pp. 35-54, 2006.

[15] K. Shafi and H. A. Abbass, "Biologically-inspired complex adaptive systems approaches to network intrusion detection," *Information Security Technical Report*, vol. 12, pp. 209-217, 2007.

[16] J. W. Begun, B. Zimmerman, and K. Dooley, "Health care organizations as complex adaptive systems," *Advances in health care organization theory*, vol. 253, p. 288, 2003.

[17] P. Nugus, K. Carroll, D. G. Hewett, A. Short, R. Forero, and J. Braithwaite, "Integrated care in the emergency department: A complex adaptive systems perspective," *Social Science & Medicine*, vol. 71, pp. 1997-2004, 2010.

[18] M. Janssen, "Use of Complex Adaptive Systems for Modeling Global Change," *Ecosystems*, vol. 1, pp. 457-463, 1998.

[19] M. A. Janssen, B. H. Walker, J. Langridge, and N. Abel, "An adaptive agent model for analysing co-evolution of management and policies in a complex rangeland system," *Ecological Modelling*, vol. 131, pp. 249-268, 2000.

[20] J. Norberg, "Biodiversity and ecosystem functioning: A complex adaptive systems approach," *Limnology and Oceanography*, vol. 49, pp. 1269-1277, 2004.

[21] C. Wycisk, B. McKelvey, and M. Hülsmann, ""“Smart parts” supply networks as complex adaptive systems: analysis and implications," *International Journal of Physical Distribution & Logistics Management*, vol. 38, pp. 108-125, 2008.

[22] C. Beckner, R. Blythe, J. Bybee, M. H. Christiansen, W. Croft, N. C. Ellis, J. Holland, J. Ke, D. Larsen-Freeman, and T. Schoenemann, "Language is a complex adaptive system: Position paper," *Language learning*, vol. 59, pp. 1-26, 2009 .

[23] E. J. Briscoe, "Language as a complex adaptive system: co-evolution of language and of the language acquisition device," 1998.

[24] S. M. Markose, "Computability and Evolutionary Complexity: Markets as Complex Adaptive Systems*," *The Economic Journal*, vol. 115, pp. F159-F192, 2005.

[25] M. J. Mauboussin, "Revisiting market efficiency: The stock market as a complex adaptive system," *Journal of Applied Corporate Finance*, vol. 14, pp. 47-55, 2002.

[26] B. B. Lichtenstein, M. Uhl-Bien, R. Marion, A. Seers, J. D. Orton, and C. Schreiber, "Complexity leadership theory: An interactive perspective on leading in complex adaptive systems," *Emergence: Complexity and Organization*, 2006 2006.

[27] M. Tilebein, "A complex adaptive systems approach to efficiency and innovation," *Kybernetes*, vol. 35, pp. 1087-1099, 2006.

[28] R. Vidgen and X. Wang, "Organizing for agility: a complex adaptive systems perspective on agile software development process," 2006.

[29] W. B. Rouse, "Health care as a complex adaptive system: implications for design and management," *Bridge-Washington-National Academy of Engineering-*, vol. 38, p. 17, 2008.

[30] R. M. Kim and S. M. Kaplan, "Interpreting socio-technical co-evolution: Applying complex adaptive systems to IS engagement," *Information Technology & People*, vol. 19, pp. 35-54, 2006.

[31] C. Rammel, S. Stagl, and H. Wilfing, "Managing complex adaptive systems — A co-evolutionary perspective on natural resource management," *Ecological Economics*, vol. 63, pp. 9-21, 2007.

[32] B. A. Cherry, "The Telecommunications Economy and Regulation as Coevolving Complex Adaptive Systems: Implications for Federalism," *Federal Communications Law Journal*, vol. 59, pp. 369-402, 2006.

[33] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *IEEE computational science and engineering*, vol. 5, pp. 46-55, 1998.

[34] L. Dalcín, R. Paz, and M. Storti, "MPI for Python," *Journal of Parallel and Distributed Computing*, vol. 65, pp. 1108-1115, 2005.

[35] *Parallel Processing with Python and OpenMP - Go Parallel*. Available: https://goparallel.sourceforge.net/parallel-processing-with-python-and-openmp/

[36] O. Dedehayir and S. J. Mäkinen, "Determining reverse salient types and evolutionary dynamics of technology systems with performance disparities," *Technology Analysis & Strategic Management*, vol. 23, pp. 1095-1114, 2011.

[37] T. P. Hughes, *Networks of power: electrification in Western society, 1880-1930*: JHU Press, 1993.

[38] S. S. Poon, R. C. Thomas, C. R. Aragon, and B. Lee, "Context-linked virtual assistants for distributed teams: an astrophysics case study," presented at the Proceedings of the 2008 ACM conference on Computer supported cooperative work, San Diego, CA, USA, 2008.