Next-Gen HPC Status Viz: Powering Up Node Status with Next.js and the Slurm API

Johnathan Lee
Research Computing
Arizona State University
Tempe, U.S.A.
johnathan.lee@asu.edu

Jason Yalim
Computational Research Accelerator
Arizona State University
Tempe, U.S.A.
yalim@asu.edu

***Abstract*—High-Performance Computing (HPC) is rapidly evolving, driven by technological advancements, growing user bases, and increasing demands for computational resources. This paper presents the transformation of Arizona State University's (ASU) user-facing HPC monitoring capabilities, focusing on a significantly updated cluster node status dashboard deployed on both of ASU's HPC systems. This dashboard represents a significant leap forward from a researcher-beloved legacy PHP status page with a custom MySQL SLURM cache that was retired in May of 2024, and a stopgap Plotly-powered visualization. The current dashboard leverages modern JavaScript frameworks and direct Slurm API integration to address the growing needs of ASU's diverse and expanding HPC user community. We explore the motivations behind the dashboard's development, detail its technical implementation, and discuss how it accommodates the growing complexity of HPC environments while simplifying user and administrator interactions. The resulting system enhances the overall user experience, lowers the barrier of entry to HPC, and also better positions ASU's HPC resources for future growth, while marking a significant milestone in the progression of HPC resource visualization. The source code is provided online with an open-source license [1].**

***Index Terms*—High-performance computing, high throughput computing, supercomputers, clusters, science gateways, scientific applications, user support, Open OnDemand applications, status pages, Next.js, Slurm API**

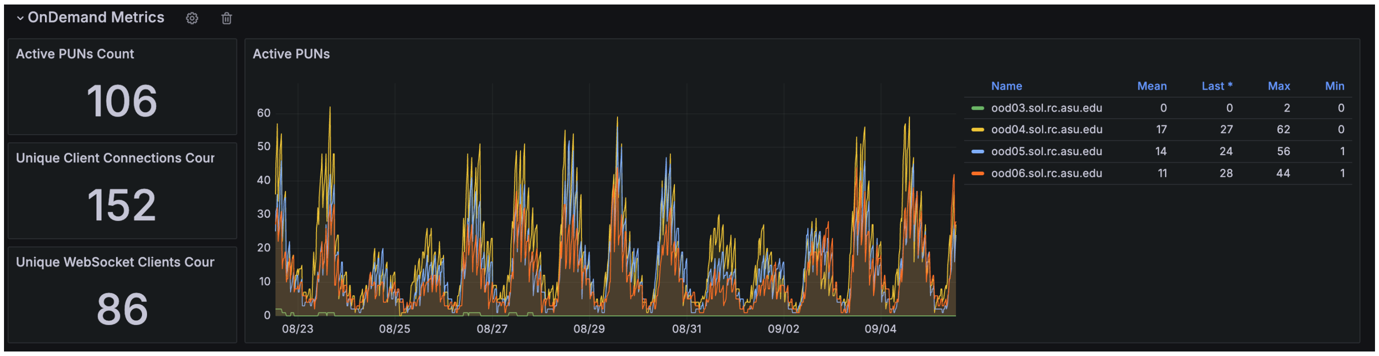
I. Introduction

Arizona State University (ASU) continues to invest in and enhance its campus high-performance computing (HPC) systems through periodic university investments. Building upon the successful implementation of the Sol supercomputer [2], ASU has recently modernized the Agave Supercomputer, now known as Phoenix. This effort physically migrated all of Agave's nodes to Sol's datacenter and upgraded the software stack to more closely match Sol's. Many of Agave's supporting systems were decommissioned. For instance, an over decade-old legacy "RCSTATUS" page which had a number of PHP- and MySQL-powered subpages for graphically viewing job scheduler data and system state. The node status page was replaced by a Plotly-powered node status page similar to Sol's [3]. However, some power users relied on pages that would allow for graphical filtering of nodes by job ID or user. This feedback prompted the creation of a next-generation dashboard.

The Plotly-powered Sol status page, which had positive attention during the RMACC23 HPC Symposium and PEARC23 conference [2], served as the core for an enhanced dashboard. While the Plotly implementation provided a high-level abstraction of the supercomputer's state and was accessible through the Open OnDemand (OOD) interface [4], the new dashboard presented here expands on these capabilities, offering increased customization and a more comprehensive view of system resources.

For the bulk of its availability, ASU's HPC offerings were only accessible to researchers with sufficient knowledge in text-based interfaces as provided by command-line terminals. Such interfaces are infamous for their steep learning curves and unforgiving leverage over HPC systems. While such interfaces will remain the most powerful way to engage in HPC tasks for the foreseeable future, the HPC learning curve has been flattening with the introduction of graphical interfaces like OSC's OOD [4]. Such platforms provide new HPC users with already familiar web-based platforms that facilitate productivity. For instance, OOD allows users without any previous command-line or HPC experience to immediately upload files, edit scripts, submit jobs to a scheduler, view job status, and view or download results. The OOD interface is similar enough to other graphical interfaces that most new HPC users need little guidance or repetitive training to complete templated tasks, which is in stark contrast with repeating the same procedures in a command line after a single formal workshop or training.

Additionally platforms like OOD provide operating system independent homogeneity — a user on Windows, MacOS, iOS, Android, and Linux all access the same set of web tools — decreasing the number of platforms that support staff have to become familiar with to onboard new, command-line adverse, users. OOD also allows users to quickly launch modern interactive applications, like Jupyter Lab servers, Rstudio, or a virtual desktop, to name a few [5], [6].



Name	Mean	Last *	Max	Min
ood03.sol.rc.asu.edu	0	0	2	0
ood04.sol.rc.asu.edu	17	27	62	0
ood05.sol.rc.asu.edu	14	24	56	1
ood06.sol.rc.asu.edu	11	28	44	1

Fig. 1. Snapshot of a Grafana dashboard monitoring instantaneous per-user NGINX (PUN) counts for the four Open OnDemand (OOD) instances dedicated to Arizona State University's main supercomputer, Sol. The green line for `ood03` is a development instance while the yellow, blue, and orange lines are production servers. The date range shown is from August 22 through September 5 in 2024, including the three-day weekend for Labor day. A peak of nearly 150 PUNs were active on August 30, demonstrating how popular OOD is among ASU researchers.

These were trends that were observed at Arizona State University (ASU) since implementing OOD in 2019. Figure 1 illustrates how popular OOD has become on ASU's Sol supercomputer, with about 75 peak users on a Labor day Monday. The success of OOD at ASU has led to the depreciation or unification of other custom web applications, like a now retired PHP and SQL powered node status page, that was in place for over a decade.

Additionally, OOD may be enhanced with site-specific utilities, like module-information or node-status pages [3]. These utilities empower all HPC users, by providing relevant system information in homogeneous, digestible formats. For instance, a node status page — an abstraction of an HPC cluster's instantaneous state by representing constituent nodes colored by their scheduler-software defined state — allows new users to develop a more concrete understanding of HPC resources, power users to increase system efficiency with greater appreciation of idleness due to heterogeneous layouts, and allows system administrators to assess with a quick glance if the HPC systems are in a critical state.

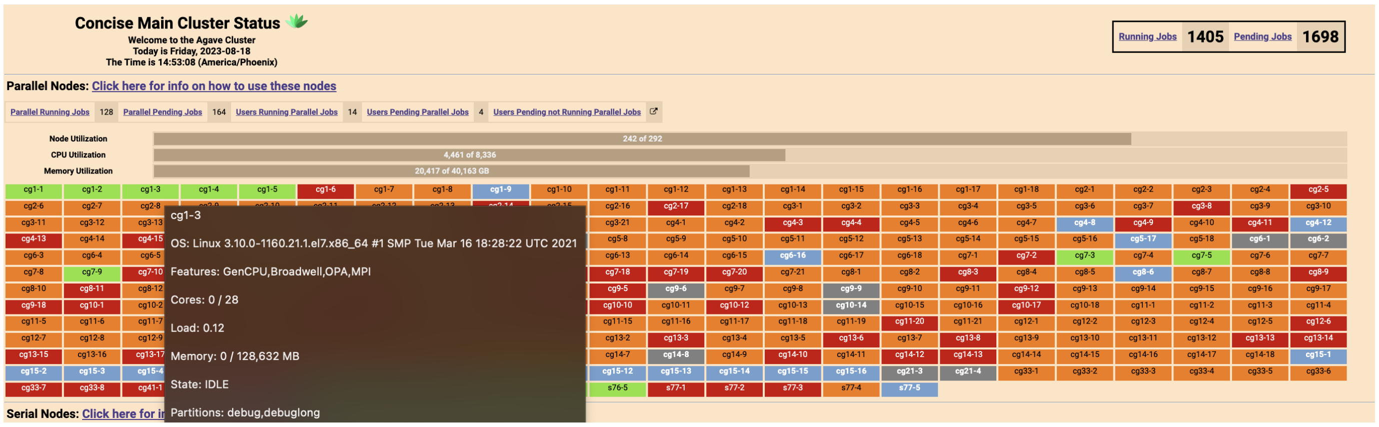
ASU Research Computing has been exclusively using the SchedMD maintained scheduling software, Slurm [7], for over a decade now. Slurm is a popular tool within the HPC community for allowing many users — that are potentially organized into arbitrary hierarchical groups (e.g., research labs) — to fairly schedule HPC work on shared computational resources [8]. The Slurm stack provides a variety of powerful command-line software tools for interfacing with the core scheduler, such as `sbatch` for submitting jobs and `squeue` for viewing the status of active jobs. However, it was not until recently that ASU became aware of Slurm's API, which forms a core basis for the flexible HPC dashboard presented here.

This paper details a significant, third-generation evolution of ASU's HPC status dashboard, served as a utility on a system's OOD page. The new stack leverages a Next.js framework and the Slurm API to provide a modern, extensible visualization tool that minimizes administrator upkeep while empowering expert users and lowers the learning curve for researchers that are new to HPC. As far as we can tell, this is the first React-based HPC node status dashboard, and we hope that the broader community will find that their users benefit from it as much as ours do. We discuss how the new dashboard not only addresses the limitations of its predecessor but also introduces innovative features that enhance the overall user experience and system management capabilities.

II. Background

Arizona State University's High-Performance Computing (ASU HPC) infrastructure has undergone a significant evolution over the last few years, and with this evolution, the researcher user base has grown substantially. While industry-standard solutions like Ganglia, InfluxDB, Prometheus, Telegraf, and Grafana form the backbone of internal system monitoring [9]–[15], they fell short of providing a user-friendly, HPC-specific status page accessible to ASU's diverse users.

Each iteration of ASU's status monitoring tools has attempted to address a set of consistent challenges. As HPC systems age, they often grow increasingly complex. This is especially true at ASU due to a "condo" model, where Faculty are able to purchase nodes that are appended to the base supercomputer. When these condo nodes are added within the first year of system genesis, they more closely resemble the base supercomputer, often having the same core, memory, and co-processor types or density (e.g., four NVIDIA A100 80GiB Graphics Processing Units). When condo nodes are purchased later in the system's life, subtle (e.g., one NVIDIA H100) or bold differences (e.g., ARM instead of x86 architecture) become increasingly apparent. For the Faculty condo "owner," these differences do not matter so much as they are provided explicit instructions for how to super-allocate their hardware. However, ASU's Sol supercomputer allows regular HPC users to compute on the nodes for up to four hours non-opportunely in a



Concise Main Cluster Status

Welcome to the Agave Cluster
Today is Friday, 2023-08-18
The Time is 14:53:08 (America/Phoenix)

Running Jobs 1405 Pending Jobs 1698

Parallel Nodes: Click here for info on how to use these nodes

Parallel Running Jobs 128 | Parallel Pending Jobs 164 | Users Running Parallel Jobs 14 | Users Pending Parallel Jobs 4 | Users Pending not Running Parallel Jobs

Node Utilization 242 of 292
CPU Utilization 4,461 of 8,336
Memory Utilization 20,417 of 40,163 GB

```
cg1-3

OS: Linux 3.10.0-1160.21.1.el7.x86_64 #1 SMP Tue Mar 16 18:28:22 UTC 2021

Features: GenCPU,Broadwell,OPA,MPI

Cores: 0 / 28

Load: 0.12

Memory: 0 / 128,632 MB

State: IDLE

Partitions: debug,debuglong
```

Serial Nodes: Click here for i

Fig. 2. Snapshot of the PHP- and MySQL-powered Research Computing node status page, "RCSTATUS," that was used for over a decade to facilitate user and system administrator demands for ASU's various HPC systems. Each color-filled rectangle represents the Slurm state of a different node on the now-obsolete Agave cluster: green boxes are completely idle, red boxes are completely allocated, orange boxes are partially allocated, blue boxes are being drained or are drained, and gray boxes are offline. The large hovertext is for Agave node `cg1-3`, a dedicated user-development node which was idle at the time of the snapshot.

"high-throughput computing" partition, or for up to a week opportunely. Power users typically seize upon idle hardware once the appropriate constraints or scheduler flags are realized. Thus, ASU's node status pages allow users to see which nodes are idle and how to access them. Also, the status page must be flexible enough to automatically grow and illustrate any new nodes and present real-time, easily digestible system information.

Another consistent challenge has been an expanding user base with varying levels of technical expertise. If ASU's HPC researchers were all experts with the scheduler's command-line interface, then a graphical node status page might only be regularly glanced at by system administrators. However, most HPC researchers are not familiar with the command-line tools that the Slurm scheduler provides, i.e., `sinfo` nor `scontrol` nor `sview` nor `s*`, and prefer to only remain familiar with allocation commands like `sbatch`, `srun`, or `salloc` and viewing the queue with `squeue`. Therefore, many researchers, even those extremely adept on HPC systems, might not know what they are missing without a node status visualization to reference.

A. *The legacy RCSTATUS system*

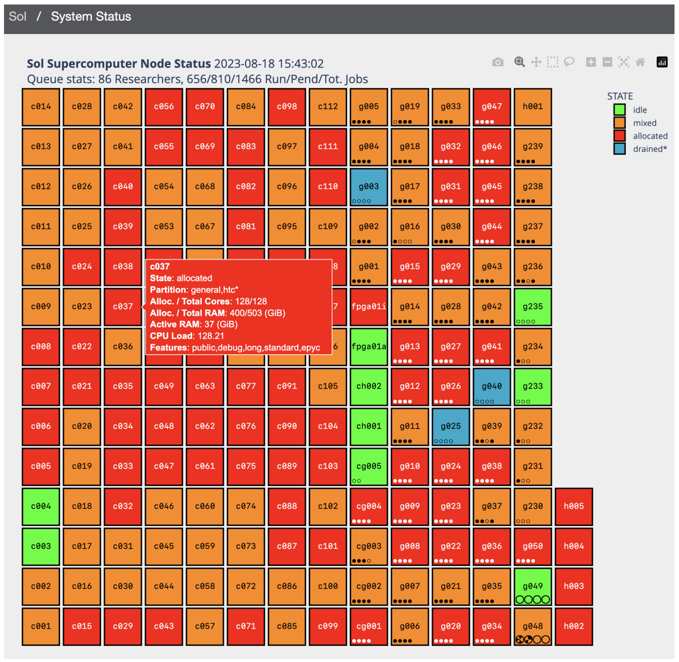
Prior to the Sol supercomputer, ASU utilized a custom HTML and PHP-based web interface, which was called "RCSTATUS." This system, running on an ASU-maintained virtual machine, queried multiple databases, including the main `slurm_acct_db` and a custom `slurm_aux_db`, to provide system and user information for every page load. While RCSTATUS offered a comprehensive view of system services and was widely used by both administrators and researchers, it faced scalability issues as the system grew, with page loading times eventually increasing to well over two minutes. This page remained popular among HPC users as it was the only graphical window into the node statuses of a previous flagship cluster, Agave. RCSTATUS was deprecated by June of 2024 with the modernization of the Agave cluster, after over ten years of continued use for several now-obsolete systems. Its node status page is illustrated in figure 2, with page partitioning of Slurm partitions ("parallel" partition in main view, "serial" partition start near the bottom of the screenshot).

B. *Plotly Based Sol Dashboard*

The introduction of the Sol supercomputer in 2021 brought with it a new Plotly-powered node status page [3], accessible through the Open OnDemand (OOD) interface. This dashboard adequately provided the features that the previous RCSTATUS node page had, with a few key improvements. As mentioned, the Plotly dashboard was accessible and effectively hosted through the OOD interface, making it significantly easier for new users to find and simplifying maintainability for system administrators (i.e., the RCSTATUS host virtual machine was retired).

The Plotly dashboard provided a more compact visual representation of the supercomputer's state, allowing all nodes to be viewed at once. It also did not rely on a custom MySQL database, instead querying the Slurm database with `sinfo` to ascertain instantaneous node state. These data were then rendered into a static HTML page as generated by Plotly's Python API, and only regenerated once a minute. Thus, this second-generation page significantly reduced the load on Sol's central databases by serving the same static webpage to everyone rather than querying the database for every page load.

The dashboard was made accessible through the supercomputer's OOD landing page, as illustrated in figure 4. The implementation required using an iframe to display the desired content, and instructions for replication were


Sol / System Status

Sol Supercomputer Node Status 2023-08-18 15:43:02
Queue stats: 86 Researchers, 656/810/1466 Run/Pend/Tot. Jobs

c037
State: allocated
Partition: general,htc*
Alloc. / Total Cores: 128/128
Alloc. / Total RAM: 400/503 (GiB)
Active RAM: 37 (GiB)
CPU Load: 128.21
Features: public,debug,long,standard,epyc

STATE
🟩 idle
🟧 mixed
🟥 allocated
🟦 drained*
c014 c028 c042 c056 c070 c084 c098 c112 g005 g019 g033 g047 h001
c013 c027 c041 c055 c069 c083 c097 c111 g004 g018 g032 g046 g239
c012 c026 c040 c054 c068 c082 c096 c110 g003 g017 g031 g045 g238
c011 c025 c039 c053 c067 c081 c095 c109 g002 g016 g030 g044 g237
c010 c024 c038 g001 g015 g029 g043 g236
c009 c023 c037 fpga01i g014 g028 g042 g235
c008 c022 c036 fpga01a g013 g027 g041 g234
c007 c021 c035 c049 c063 c077 c091 c105 ch002 g012 g026 g040 g233
c006 c020 c034 c048 c062 c076 c090 c104 ch001 g011 g025 g039 g232
c005 c019 c033 c047 c061 c075 c089 c103 cg005 g010 g024 g038 g231
c004 c018 c032 c046 c060 c074 c088 c102 cg004 g009 g023 g037 g230 h005
c003 c017 c031 c045 c059 c073 c087 c101 cg003 g008 g022 g036 g050 h004
c002 c016 c030 c044 c058 c072 c086 c100 cg002 g007 g021 g035 g049 h003
c001 c015 c029 c043 c057 c071 c085 c099 cg001 g006 g020 g034 g048 h002

Fig. 3. Snapshot of the Plotly-powered Sol status page, that was rapidly deployed to satisfy ASU's HPC user community in 2021. Like the previous node status page, color-filled boxes are used to abstract Sol's nodes and their statuses, and empty or filled circle-markers annotate graphics processing unit (GPU) nodes' lower left corner to indicate GPU allocation status. A hovertext for Sol node `c037` is also shown. Additional details are in the conference paper [3].

included in the supplemental open source repository [3], [16].

Finally, the implementation significantly minimized the quantity of in-house software to maintain, thus also reducing the burden on administrators. This did not restrict new code development: a custom Python method was written to draw pie-chart representations of multi-instance GPU (NVIDIA MIG) allocations, as seen in the bottom row, second-to-last column for node `g048` in figure 3.

In addition to researcher satisfaction, the positive reception of the Sol dashboard at conferences and among peers highlighted the demand for such tools in the HPC community. While this second-generation Sol dashboard significantly improved upon RCSTATUS and was fast to deploy from scratch for Sol, it was not intended to be a long-term solution and had clear areas for potential enhancement. The dashboard was a simple illustration of node status, using only simple javascript-enabled widgets to provide the hovertext or interactive legend. However, slightly more javascript (beyond the Plotly.express used here) could easily allow for richer features, like custom filters of the nodes, which was requested by several researchers that "own" hardware. Additional feedback from users and administrators, along with evolving system requirements, identified several areas for improvement:

1) Custom filters, like user name or job ID, as well as dropdown menus for common filters, like Slurm partition names or node features.
2) Custom actions on node click, like sub-windows showing node load history and current jobs.
3) Enhanced customization options, like different card sizes, and levels of detail, to cater to different user roles and preferences.
4) Using the Slurm API for more efficient data retrieval and processing to support larger-scale systems.
5) Improved visualization of complex resources, detailed job status, and system overview details.
6) Build on top of existing tools, like Prometheus, InfluxDB, and the Slurm API.
7) Support for user-facing historical Slurm or node analysis.
8) Support for displaying other useful details, such as software module information.

These factors, combined with the continuous growth of ASU's HPC infrastructure, motivated the development of a new, long-term dashboard that would not only address these needs but also set a new standard for HPC system monitoring and visualization tools.

III. New dashboard implementation with Next.js

Building upon the foundations laid by the second-generation HPC node status dashboard [3], the new dashboard represents a significant leap forward in both technology and user experience. This section details the key improvements and design philosophy behind the new implementation.

A. Technology Stack

The new dashboard leverages modern web technologies to provide a standardized, efficient, and user-friendly interface. A diagram of the new status page's technology stack is provided in figure 5. The dashboard is built using Next.js, a React-based javascript framework known for its performance optimization and server-side rendering capabilities. This choice allows for faster load times and improved overall responsiveness compared to the previous implementation. Next.js is well-documented online and was created and is maintained by the company, Vercel [17].

The dashboard also leverages the Slurm API. Direct integration with the Slurm [7] API enables quick, standardized response data, providing users with the most current system information without the need for manual or HTML-dependent page refreshes. The original RCSTATUS page depended on a custom MySQL database of Slurm data that had significant performance degradation by the end of its life. The Ploty-powered status page improved on this by querying the Slurm database with `sinfo` queries, but these were done once a minute and cached into the static HTML page. These `sinfo` queries were quirky, for instance, fields had to be sufficiently wide to prevent message truncation. These previous generations would both refresh their content by using the HTML `meta` tag's `http-equiv="refresh"` attribute.

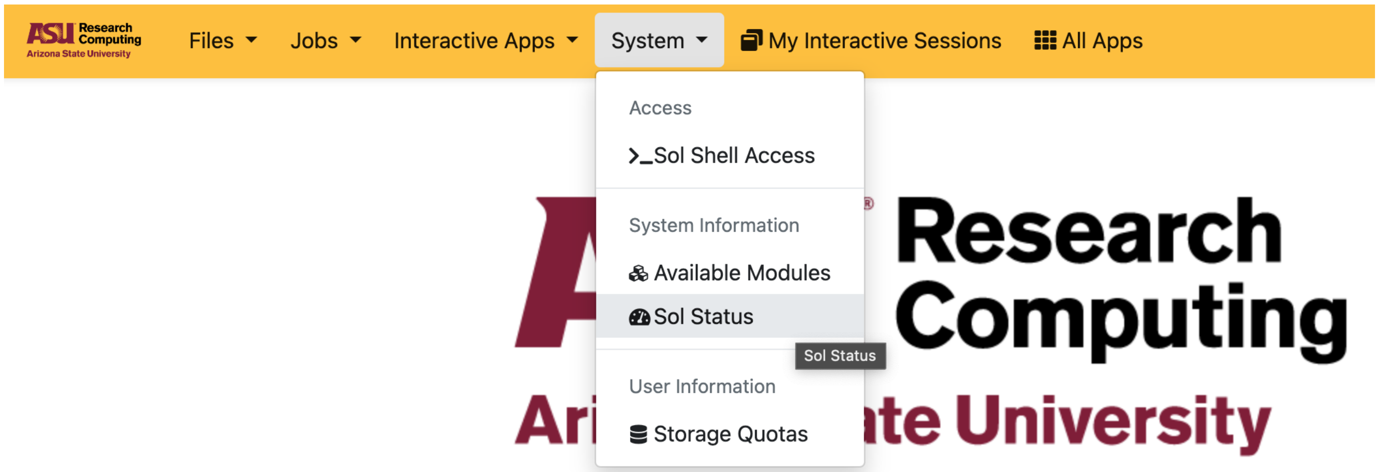


Fig. 4. Snapshot of Sol's Open OnDemand (OOD) landing page, with the "System" top-page navigation bar drop-down menu activated. From the drop down under "System Information", "Sol Status" is highlighted. Clicking on "Sol Status" takes the end user to a subpage of OOD that provides an HTML iframe view of Sol's status page.

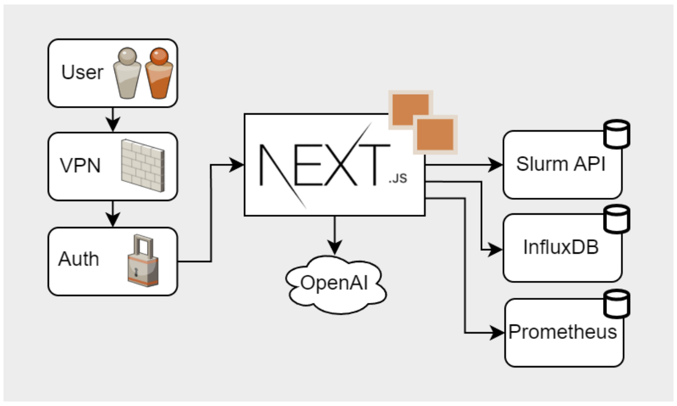


Fig. 5. Diagram of the Next.js-powered node status page. Users that are verified first by the University virtual private network and then authenticated through the University's single sign-on, have access to ASU's supercomputer and the dashboard. By connecting to the Open OnDemand (OOD) interfaces, users will gain access to the Next.js powered dashboard (as illustrated in figure 4). The dashboard is powered by the Slurm API, InfluxDB, and Prometheus, and can be configured to work via API-key to OpenAI and ChatGPT.

This technology stack addresses several limitations of the previous dashboard, specifically in terms of available data and how those data are presented.

B. Design Philosophy

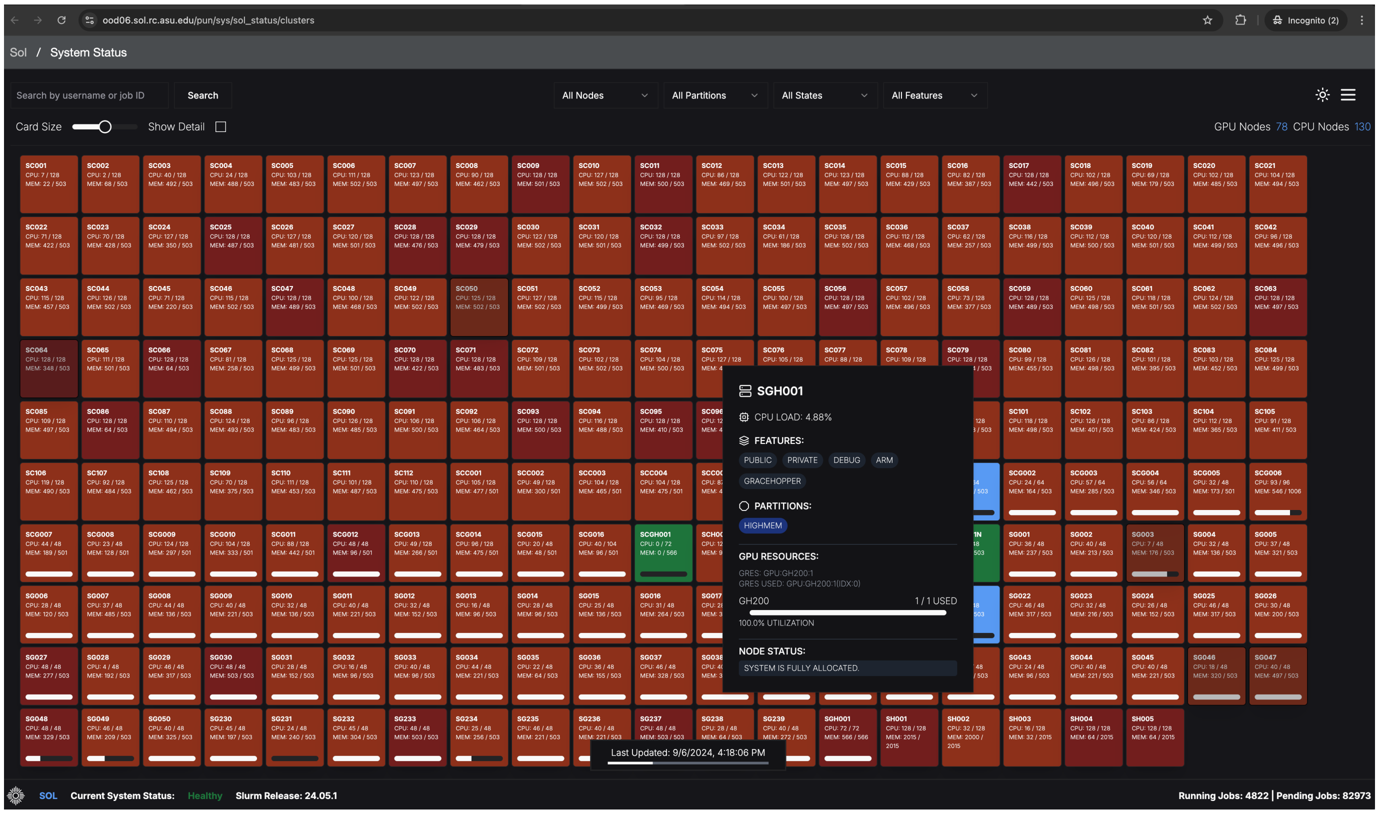
The Next.js-powered dashboard was developed with several key principles in mind, aimed at improving both user experience and system maintainability:

1) **Sleek and User-Friendly Interface**: The dashboard features a modern, intuitive design that prioritizes ease of use. Visual elements are built to present complex system information in an easily digestible format.
2) **Easy Maintenance**: The code base is structured for straightforward updates and modifications. This design choice ensures that the dashboard can evolve alongside the HPC system, with minimal administrative overhead.
3) **Simple Deployment**: The dashboard is designed for easy installation and configuration across different HPC environments. This "easy to stand up, get up and running in five minutes" approach also makes it accessible to a wider range of institutions and reduces the barrier to adoption.
4) **Customization and Extensibility**: Building on the customization options introduced in the previous version, the new dashboard offers enhanced flexibility. Administrators and users can easily modify the interface and add new features to suit their specific needs.

C. Key Improvements

The new dashboard introduces several enhancements over its predecessors; the landing page is illustrated in figure 6. Leveraging the Slurm API and Next.js's efficient rendering, data are updated every thirty seconds. Due to the extensibility of Next.js and the data availability, work is also in progress to build a full job history toolkit, to help users better understand why their jobs may have failed or how to tune allocations in what is envisioned as a web `sacct` and `seff`. Towards this end, if a user supplies an inactive job ID in the upper left search box in figure 6, a dialog like figure 8 will populate with details about that job and a simple percentage and associated letter-grade score for the job's CPU and memory efficiencies. If an active job ID is instead requested, a simpler dialog is populated with details similar to that of `scontrol show job <ID>`.

The new dashboard offers enhanced visualization of the HPC system's current and historical state. Improved graphics and interactive elements offer a more detailed and


ood06.sol.rc.asu.edu/pun/sys/sol_status/clusters
Sol / System Status
Search by username or job ID
Search
All Nodes
All Partitions
All States
All Features
Card Size
Show Detail
GPU Nodes 78 CPU Nodes 130

SGH001
CPU LOAD: 4.88%
FEATURES: PUBLIC PRIVATE DEBUG ARM GRACEHOPPER
PARTITIONS: HIGHMEM
GPU RESOURCES:
GRES: GPU:GH200:1
GRES USED: GPU:GH200:1(IDX:0)
GH200 1 / 1 USED
100.0% UTILIZATION
NODE STATUS:
SYSTEM IS FULLY ALLOCATED.

Last Updated: 9/6/2024, 4:18:06 PM

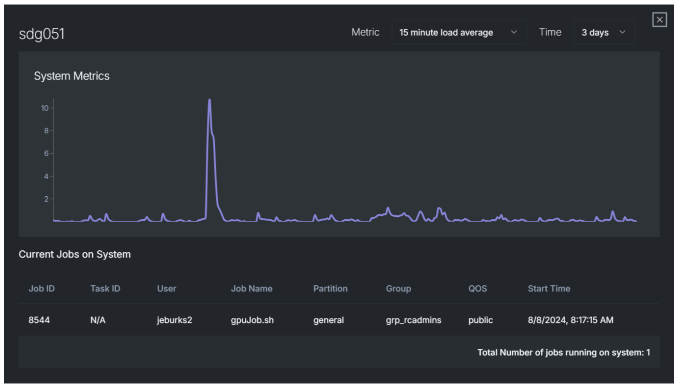
SOL Current System Status: Healthy Slurm Release: 24.05.1 Running Jobs: 4822 | Pending Jobs: 82973

Fig. 6. Snapshot of the Next.js-powered node status page, demonstrated for the Sol supercomputer. Sol's nodes are represented by color-filled boxes, with color corresponding to the node's Slurm state. Green, orange, and red indicate nodes that are idle, partially allocated, and fully-allocated, respectively. Gray indicates a down node, blue, a drained node, and indigo, a node reserved for maintenance. When the CPU load average on a node is in excess of a normalized 100%, the node's color will begin to gently pulse, helping alert system administrators and users to poorly-threaded compute jobs. Nodes `sc050`, `sc064`, `sg003`, `sg046`, and `sg047` are in this overloaded state, and appear slightly darker as a result. Clicking on a node would result in a new dialog as illustrated by figure 7, showing which jobs are running with standard queue information. The text on each box displays a node's hostname and the number of cores, and the memory (in GiB). Allocated GPUs are indicated with a light continuous slider. There are four dropdown menus at the top-center of the page, allowing for custom filtering from defined fields. A search bar in the upper-left allows for further filtering for users or job numbers, the latter opens a new dialog like figure 8, which includes job efficiencies for no longer active jobs. Just underneath that search is a slider for "card size," which controls the node's pixel area. The page shows the time it was last updated and the time to the next update in a gray box at the bottom-center of the page. Some simple queue statistics are shown in the bottom-right corner. A hovertext is active for node `sgh001`, displaying configuration and allocation details. The glowing-light icon in the upper-right (to the left of the burger) toggles "dark mode," adjusting the background color. The top address bar reflects that this is live on an Open OnDemand server, hosted as an iframe [16].

intuitive representation of system resources. Figure 7 is a cropped screenshot of the sub-window that appears when clicking on a node, in this case, Sol's `sdg051`. The sub-window smoothly builds a time series plot of the node's historical CPU load over one week as well as a table of the node's active jobs. The node detail view in figure 7 illustrates completely new features made easily available by the Next.js framework as well as how easy it can be to add secondary options, like different series to visualize and different time windows. Improved visualizations improve user onboarding experiences by decreasing the abstraction of remote supercomputing.

The dashboard is fully responsive, providing an optimal viewing experience across a wide range of screen sizes, which makes it interpretable for a wide range of devices or browser window sizes. This is a natural benefit of using a modern web-framework like Next.js — decades of practical web development is inherited and nearly automatic at project initialization. For instance, clicking on a node results in a simple animation building the view like in figure 7. This initial view has to actually request or load the data, and the animation helps smooth a potentially lagged transition. However, the view is cached and rebuilt more quickly if requested again later. This fundamental combination of form and function is a consistent and welcome Next.js theme.

The flexibility of the Next.js framework allows for richer supporting content. Figure 10 illustrates an ability to recall a historical node state. Figure 9 is a screenshot of a software module table, a supplemental component of the main dashboard. Sol already had an existing, OOD linked, DataTables.js-powered static HTML viewer for software modules. This Next.js implementation allows the end user to remain on the dashboard to check modules, and as

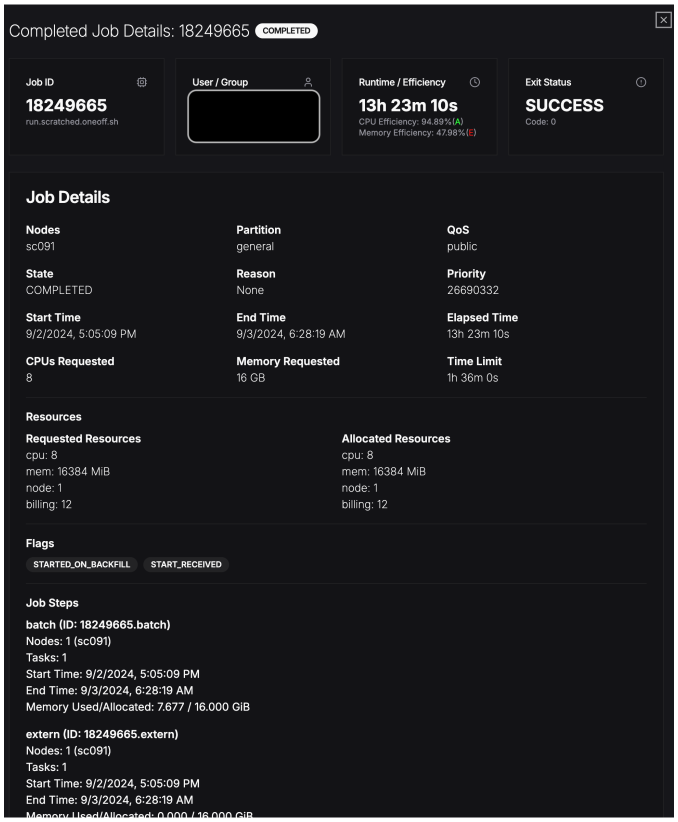


Current Jobs on System

Job ID	Task ID	User	Job Name	Partition	Group	QOS	Start Time
8544	N/A	jeburks2	gpuJob.sh	general	grp_rcadmins	public	8/8/2024, 8:17:15 AM

Total Number of jobs running on system: 1

Fig. 7. Cropped screenshot of the node display sub-window for Sol's `sdg051` node. The top of the sub-window plots the node's three-day historical CPU load. The bottom of the sub-window presents the node's currently running jobs in a scrollable table, via a Slurm API query. To return to the main status page, an end-user may click the "x" in the upper-right of the sub-window or click on the background node status page. The historical CPU load data are loaded from Prometheus. Also illustrated are features that allow for different time series plots over different time windows. A fifteen minute CPU load average over three days is being plotted here. Full options include five or fifteen minute load averages, memory usage, and a cumulative sum of out-of-memory kills, all shown over either one, three, or seven days.



Completed Job Details: 18249665 COMPLETED

Job ID
18249665
run.scratched.oneoff.sh

User / Group
[redacted]

Runtime / Efficiency
13h 23m 10s
CPU Efficiency: 94.89% (A)
Memory Efficiency: 47.98% (E)

Exit Status
SUCCESS
Code: 0

Job Details

Nodes sc091	**Partition** general	**QoS** public
State COMPLETED	**Reason** None	**Priority** 26690332
Start Time 9/2/2024, 5:05:09 PM	**End Time** 9/3/2024, 6:28:19 AM	**Elapsed Time** 13h 23m 10s
CPUs Requested 8	**Memory Requested** 16 GB	**Time Limit** 1h 36m 0s

Resources

Requested Resources	Allocated Resources
cpu: 8	cpu: 8
mem: 16384 MiB	mem: 16384 MiB
node: 1	node: 1
billing: 12	billing: 12

Flags
STARTED_ON_BACKFILL START_RECEIVED

Job Steps

batch (ID: 18249665.batch)
Nodes: 1 (sc091)
Tasks: 1
Start Time: 9/2/2024, 5:05:09 PM
End Time: 9/3/2024, 6:28:19 AM
Memory Used/Allocated: 7.677 / 16.000 GiB

extern (ID: 18249665.extern)
Nodes: 1 (sc091)
Tasks: 1
Start Time: 9/2/2024, 5:05:09 PM
End Time: 9/3/2024, 6:28:19 AM
Memory Used/Allocated: 0.000 / 16.000 GiB

Fig. 8. Cropped screenshot of the details for queried job, 18 249 665, with the user and group names redacted. The Slurm API was queried and scraped for overall CPU and memory efficiencies, which are scored from a letter-grade scale in the "Runtime/Efficiency" box. The memory utilization details of every job step are included at the bottom of the window, after which the working directory and submit command are displayed (not illustrated).

figure 9 illustrates, to see the module's description (as defined in the corresponding modulefile) as a hovertext. This ability has lead to an internal project to determine a custom modulefile schema to improve researcher and administrator quality of life.

Another key improvement of the Next.js dashboard is how well it facilitates integration with other well-known tools and services, like InfluxDB, Prometheus, or OpenAI. Figures 7 and 10 build their historical load average time series from Prometheus. Figure 11 uses Vercel's AI-SDK [18] to integrate OpenAI, building wrapped Slurm API queries based on user interaction.

By addressing the limitations identified in the previous implementation [3], listening to the provided feedback, and incorporating modern web technologies, this Next.js-powered dashboard sets a new standard for HPC system monitoring tools. Its combination of powerful features, user-friendly design, and easy maintainability makes it an invaluable asset for both system administrators and researchers in the community.

D. *Installation and Configuration*

The HPC dashboard has been designed with ease of installation and flexibility in mind, with various deployment methods and optional integrations. This section provides an overview of the installation process and configuration options.

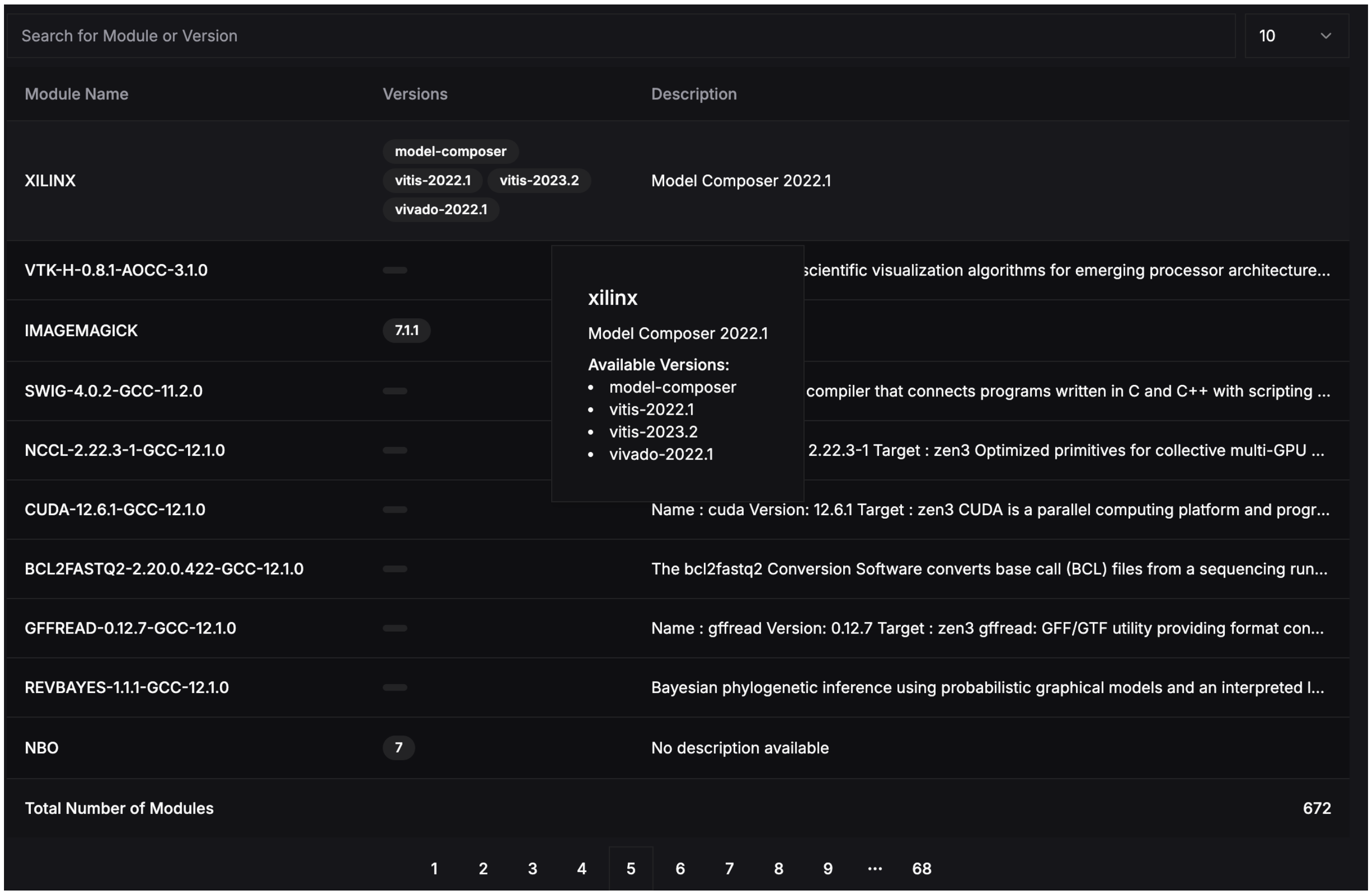
1) Prerequisites: Before beginning the installation, users should ensure they have:

- **Required:** Slurm API access and a valid API key, Node.js, and `npm`.
- **Optional:** Prometheus endpoint, InfluxDB instance, OpenAI API key.

2) Installation Overview: The installation process can be summarized in a few key steps:

1) Clone the repository from GitHub.
2) Install dependencies using `npm`.
3) Configure the environment file with the necessary API key.
4) Start the development server.

For detailed, step-by-step instructions, users are encouraged to refer to the README file in the supplemental GitHub repository [1]. For production deployments or to leverage additional features, administrators will likely want to use a more robust solution, i.e., ensuring that the system is available over HTTPS and starts when the system reboots. Detailed guides for these advanced configurations (for the web server, SSL/TLS for HTTPS, process management with tools like PM2, and integration with the optional services [Prometheus, InfluxDB, and OpenAI]) are available in the documentation section of the GitHub repository. Note that the supplemental GitHub repository also provides instructions for connecting the dashboard service to a production Open OnDemand instance.



Search for Module or Version

Module Name	Versions	Description
XILINX	model-composer vitis-2022.1 vitis-2023.2 vivado-2022.1	Model Composer 2022.1
VTK-H-0.8.1-AOCC-3.1.0		scientific visualization algorithms for emerging processor architecture...
IMAGEMAGICK	7.1.1	
SWIG-4.0.2-GCC-11.2.0		compiler that connects programs written in C and C++ with scripting ...
NCCL-2.22.3-1-GCC-12.1.0		2.22.3-1 Target : zen3 Optimized primitives for collective multi-GPU ...
CUDA-12.6.1-GCC-12.1.0		Name : cuda Version: 12.6.1 Target : zen3 CUDA is a parallel computing platform and progr...
BCL2FASTQ2-2.20.0.422-GCC-12.1.0		The bcl2fastq2 Conversion Software converts base call (BCL) files from a sequencing run...
GFFREAD-0.12.7-GCC-12.1.0		Name : gffread Version: 0.12.7 Target : zen3 gffread: GFF/GTF utility providing format con...
REVBAYES-1.1.1-GCC-12.1.0		Bayesian phylogenetic inference using probabilistic graphical models and an interpreted l...
NBO	7	No description available
Total Number of Modules		672

xilinx
Model Composer 2022.1
Available Versions:
- model-composer
- vitis-2022.1
- vitis-2023.2
- vivado-2022.1

Fig. 9. Screenshot of the software modules table from the Sol supercomputer's Next.js-powered dashboard. Hovering over a row allows for easier reading of the module's description. The module names, versions, and descriptions are pulled via a daily cron job that saves `lmod` result to JSON. The app has a dynamically filtering search at the top of the window.

IV. Future Work

The HPC dashboard is an evolving project, with ongoing development driven by user feedback and emerging technologies in the HPC landscape. This section outlines the planned enhancements and potential areas for community contribution.

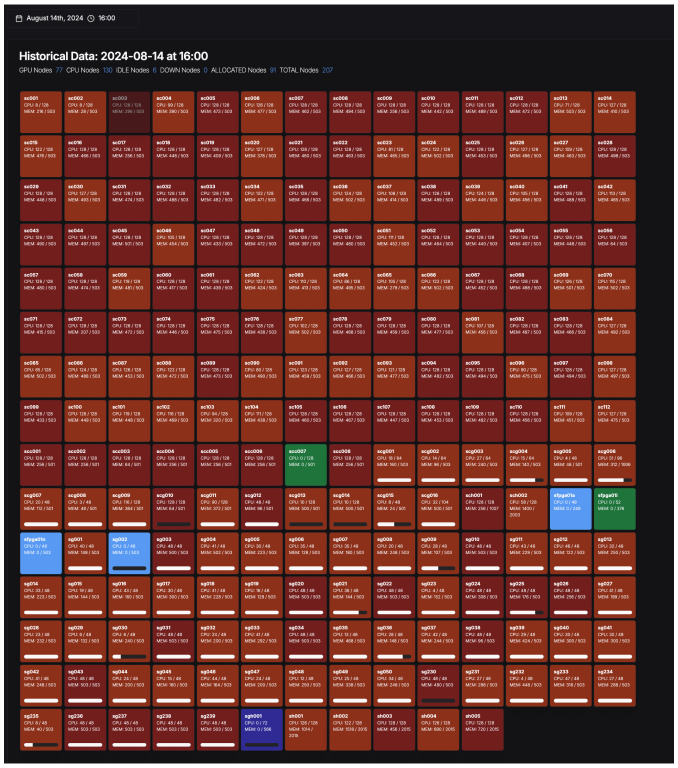
Planned Enhancements

Slurm Job Debugging and Efficiency reports. Like the Slurm command-line tools `sacct` or `seff`, the Slurm API can access detailed job information for any valid job. While the current status page provides `seff`-like scoring of CPU and memory efficiency — as demonstrated by figure 8 — a natural supplemental dashboard component would allow a user to query multiple jobs. While ASU employs XDMoD [19], the XDMoD database implementation does not update at the same pace as Slurm "forgets" recently completed jobs and is a much more serious and statistical tool for simple tasks that the proposed dashboard component may solve simply.

Advanced Customization and Accessibility Features. The current color mapping for the node status page often results in green (idle) and red (fully allocated) nodes adjacent to one another. This mapping has been in place since the first-generation RCSTATUS page, but may not be the most accessible, especially for those with red–green colorblindness. To address this, a screenshot of the status page was taken and passed to the program `peacock` [20], which created simulations of the page view for those with protanopia, protanomaly, deuteranopia, deuteranomaly, tritanopia, and tritanomaly. While sufficient contrast between the node states resulted in these simulated views, it is not ideal to enforce a single color mapping on the general community. Thus, one planned feature will be to allow for custom color maps.

Slurm Chat Enhancements. The current Slurm Chat functionality, while promising, relies on proprietary APIs. Some of the directions include investigating in-browser large language model capabilities for client-side processing or adding the ability to use local embeddings, to bring in site documentation for an even better end-user experience.

Admin Console Development. To streamline setup and management, it would be desirable to have an admin console that implements role-based access control (RBAC) for granular permission management. This console would ideally also allow administrators to configure the dash-

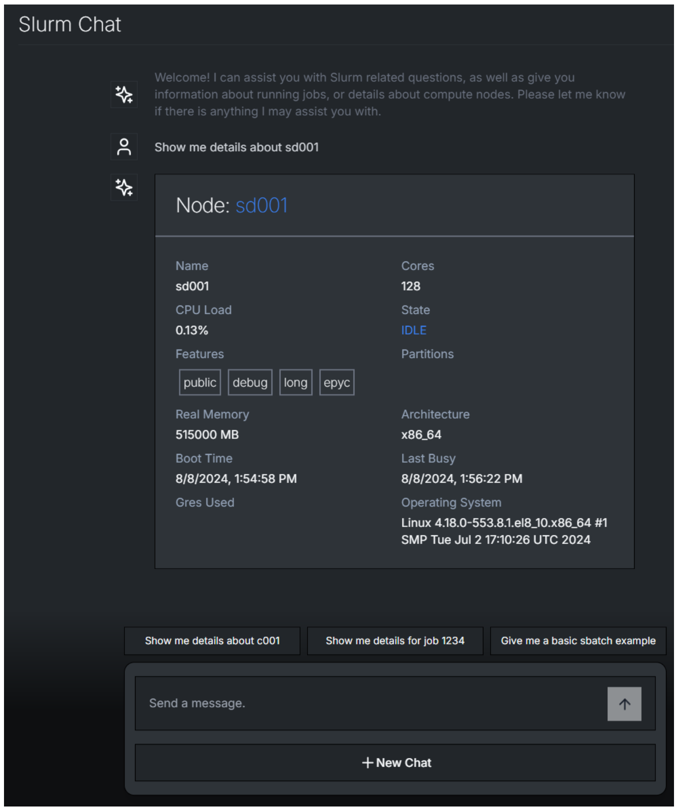


August 14th, 2024 16:00

Historical Data: 2024-08-14 at 16:00

GPU Nodes 77 CPU Nodes 130 IDLE Nodes 6 DOWN Nodes 0 ALLOCATED Nodes 91 TOTAL Nodes 207

Fig. 10. Screenshot of Sol's Next.js-powered dashboard, illustrating "historical mode" for Sol on August 14, 2024 at 4 PM M.S.T.. The historical data are loaded from JSON as requested by the date and time widget in the upper-left of the page.



Slurm Chat

Welcome! I can assist you with Slurm related questions, as well as give you information about running jobs, or details about compute nodes. Please let me know if there is anything I may assist you with.

Show me details about sd001

Node: sd001

Name	Cores
sd001	128
CPU Load	State
0.13%	IDLE
Features	Partitions
public debug long epyc	
Real Memory	Architecture
515000 MB	x86_64
Boot Time	Last Busy
8/8/2024, 1:54:58 PM	8/8/2024, 1:56:22 PM
Gres Used	Operating System
	Linux 4.18.0-553.8.1.el8_10.x86_64 #1 SMP Tue Jul 2 17:10:26 UTC 2024

Show me details about c001
Show me details for job 1234
Give me a basic sbatch example
Send a message.
+ New Chat

Fig. 11. Cropped screenshot of Sol's development Next.js-powered dashboard, illustrating an OpenAI-powered ChatGPT chat application. The chat shows a custom help text at the start, prompting the user on how to use the bot to query Sol Slurm information. The user then requested "Show me details about sd001," which led to a formatted table of that node's state, as reported by Slurm. This was done with Vercel's AI-SDK, allowing for custom actions, like querying a wrapped Slurm API for node information based on the user's prompt.

board components, data sources, and visualization options through the browser.

Advanced Visualization and Analytics. Enhancing the dashboard's ability to represent complex HPC environments by developing real-time network topology visualizations to illustrate inter-node communication patterns and creating customizable dashboards for power consumption and thermal overviews.

V. Conclusion

The background, motivation, and methodology for a Next.js-powered high-performance computing (HPC) status dashboard were presented. The dashboard is a third-generation web app from the Arizona State University (ASU) HPC team, reflecting significant improvements in its fundamental design and the likely longevity of the service. The dashboard is accessible through Open OnDemand, serves an orthogonal role to the user-facing XDMoD, and orthogonal roles to more powerful monitoring tools like Telegraf, Prometheus, InfluxDB, and Grafana, all of which are actively utilized for things like node load, heat, and power draw as well as software module loading.

The dashboard's core is a node status page which leverages the Slurm API to determine and compactly illustrate the instantaneous state of the system's nodes. The nodes are abstracted as color-filled squares and the colors indicate Slurm status. The default square size is large enough for the annotation of CPU cores, memory, and GPU allocation ratios. As a result, end users of any skill level may ascertain the availability of computational resources with a quick glance. Additional details may be obtained by hovering over or clicking on nodes, in the latter case providing information like the historical CPU load average over the last week and a list of running jobs. The dashboard also allows for quick filtering for nodes based on a job number or a username, as well as by Slurm attributes like a partition or node features.

The Next.js framework has led to the rapid development of a number of features and supporting components. For instance, an interactive table of system software modules was developed, which is compiled from the system's modulefiles. Additionally, an OpenAI-powered chat interface was created that allows users to query a wrapped Slurm API by interacting with the bot.

Given the internal popularity of the first-generation status pages, and the general popularity of the second-generation node status page, we hope that the external community will again appreciate this modern third-generation node status page. The source code is available on Github [1] and collaborations are welcome.

Acknowledgements

We would like to thank Lee Reynolds for creating the initial "RCSTATUS" PHP-based page, as well as the Arizona State University high-performance computing research community and the rest of the Research Computing team for their valuable feedback on the design.

References

[1] Johnathan Lee. Slurm node dashboard, 2024. https://github.com/thediymaker/slurm-node-dashboard.

[2] Douglas M. Jennewein, Johnathan Lee, Chris Kurtz, Will Dizon, Ian Shaeffer, Alan Chapman, Alejandro Chiquete, Josh Burks, Amber Carlson, Natalie Mason, Arhat Kobwala, Thirugnanam Jagadeesan, Praful Barghav, Torey Battelle, Rebecca Belshe, Debra McCaffrey, Marisa Brazil, Chaitanya Inumella, Kirby Kuznia, Jade Buzinski, Sean Dudley, Dhruvil Shah, Gil Speyer, and Jason Yalim. The Sol Supercomputer at Arizona State University. In *Practice and Experience in Advanced Research Computing*, PEARC '23, New York, NY, USA, 2023. Association for Computing Machinery.

[3] Jason Yalim. Visualizing supercomputer node status on Open OnDemand with Plotly. In *Rocky Mountain Advanced Computing Consortium*, RMACC HPC Symposium 2024. Zenodo, 2024.

[4] Dave Hudak, Doug Johnson, Alan Chalker, and et al. Open ondemand: A web-based client portal for hpc centers. *J. of Open Source Software*, 3(25):622, 2018.

[5] Jupyter Lab. Jupyter lab, 2024. https://github.com/jupyterlab/jupyterlab.

[6] Rstudio. Rstudio, 2024. https://github.com/rstudio/rstudio.

[7] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 44–60, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[8] Jason Yalim. Toward dynamically controlling slurm's classic fairshare algorithm. In *Practice and Experience in Advanced Research Computing 2020: Catch the Wave*, PEARC '20, pages 538–542, New York, NY, USA, 2020. Association for Computing Machinery.

[9] Prometheus. Prometheus, 2024. https://github.com/prometheus/prometheus.

[10] Nitin Sukhija and Elizabeth Bautista. Towards a framework for monitoring and analyzing high performance computing environments using kubernetes and prometheus. In *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 257–262, 2019.

[11] Influxdata. Telegraf, 2024. https://github.com/influxdata/telegraf.

[12] Grafana. Grafana, 2024. https://github.com/grafana/grafana.

[13] Ganglia. Ganglia, 2024. https://github.com/ganglia.

[14] Influxdata. Influxdb, 2024. https://github.com/influxdata/influxdb.

[15] Jie Li, Ghazanfar Ali, Ngan Nguyen, Jon Hass, Alan Sill, Tommy Dang, and Yong Chen. Monster: An out-of-the-box monitoring tool for high performance computing systems. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 119–129, 2020.

[16] Johnathan Lee. Open ondemand status html iframe, 2024. https://github.com/thediymaker/ood-status-iframe.

[17] Vercel. Next.js, 2024. https://github.com/vercel/next.js.

[18] Vercel. Ai-sdk, 2024. https://github.com/vercel/ai.

[19] Jeffrey T. Palmer, Steven M. Gallo, Thomas R. Furlani, and et al. Open xdmod: A tool for the comprehensive management of high-performance computing resources. *Computing in Science & Engineering*, 17(4):52–62, 2015.

[20] Joel A. Kulesza. Peacock, 2021. https://github.com/jkulesza/peacock.