Future of Computation - Needs and Trends

or An Idiosyncratic Airing of Grievances, Condor-Style

Peter Couvares, LIGO Laboratory / Caltech

LIGO-G2200862-v1

Who am I?

Why Am I Here?

- Started in industry (Consulting, Telecom, Medical) ’90-’99.
- Early staff member of the HTCondor Team (’99-’08).
- Back to industry (Data Visualization) ’08-’09.
- Research Scientist @ Syracuse University Physics Dept. (’10-’15) focused on distributed computing problems for the LIGO Scientific Collaboration (LSC), and building research computing capabilities at Syracuse.
- Scientist in the LIGO Laboratory @ Caltech (’15-today), managing LIGO data analysis computing, and leading the LSC computing optimization effort (broadly defined).
- **I’m here because (1) HTCondor powers LIGO Science, (2) LIGO appreciates and values this community <u>enormously</u>, (3) and I love and miss Madison!**

What’s Hard (1/5)

Resource Planning & Scheduling

- Increasingly heterogenous resource demands — tricky for capacity planning and scheduling
    - “low-latency/online” HTC CPU demand (time-sensitive work — has to be run immediately or not at all)
    - “batch/offline” HTC CPU demand
    - HTC GPU demand
    - H***P***C CPU (MPI) demand
- Balancing “low-latency/online” vs “batch/offline” HTC CPU demand
    - circa 2004: Bologna Batch System
    - O1-O2 (2015-2017): dedicated CPUs (same pool, start policy runs low-latency/online jobs only) — often underutilized
    - During O3 (2019-2020): dual startds (one online, one batch), running with different cgroup settings.
    - New for O4: single startd managing multiple partitionable slots (online vs batch) with separate cgroup settings.
    - Works well, but issues include: static partitioning of memory between high- and low-priority jobs whether or not they are both running — we either underutilize RAM or risk overcommitment.

What’s Hard (2/5)

HTC CPU Prioritization

- LIGO/Virgo/KAGRA supply and demand very difficult to predict
  - ~500 active users working on ~80 distinct scientific projects (aka “searches”), all on their own dev/test/sim/prod + paper-writing schedule
  - total need over 3 years: we now have a very good handle on this!
  - timing of need during those years: no idea! life comes at you fast…
- Our default is bog-standard Condor fair-share scheduling by user.
- Lots of reactive, manual user priority tweaking — when an urgent science goal needs to get done faster, we adjust user priorities.
- But unix users != scientific projects, and priority tweaks need to be un-done when urgency is over.
  - Note: new user-priority ceiling very helpful — we can jack up someone’s priority without worrying about them starving others
- What do we want? We don’t know exactly (sorry Miron, no poetry for you!) — but:
  - less manual tweaking of knobs of that are poor proxies for what we care about (relative weighting of competing large projects and deadlines on one hand, avoiding individual user starvation on the other)
  - to be able to reason about projects and deadlines, rather than (or in addition to?) unix users and fair-share priorities

What’s Hard (3/5)

Grid Flocking

- E.g., a user on a LIGO Access Point (AP) at Caltech might have access to:
  - IGWN Grid
  - Local Condor pool
  - Open Science Pool
- Painful UX
  - HTCondor command-line tools can be told to connect to a (single) different HTCondor Central Manager hostname, but they don’t understand or represent a multi-pool model at all.
- Very difficult debugging
  - Different grid pilot environments — even when containerized — lead to user-impacting differences in job behavior. Grid flocking == mayhem.
  - Users cannot easily tell which CM a given EP was provided by.

What's Hard (4/5)

Grid "Sites"

- Yes, I know it's an ill-defined concept, but it is a **critically** useful one, which we absolutely must regularly use in practice.
- One working definition: a set of colocated Execution Points (EPs) under common administrative control with a homogenous runtime environment *on which user-impacting failures will tend to correlate* (i.e., they all go offline and fail together, they all exhibit the same job dysfunction due to their common configuration, they all experience slow data xfer due to their common LAN, etc.)
- **Painful UX — HTCondor has no concept of sites at all, but admins and users need to reason about them constantly.**
- Users often want to target a specific site (for all kinds of reasons — testing, debugging, to use an allocation, etc.), and cannot easily do so.
- It is very difficult for users (or AP admins) to recognize that a series of job failures all involve a common site. But it happens all. the. time.
- Users (or AP admins) cannot easily deny-list a site, even though this is often the single most logical and efficient way to work around an infrastructure problem you can't solve.
- When users (or AP admins) do manage to block a site (or get their grid admin to do it at the factory level), they have no way of knowing when it's working again, and often leave it disabled long after the problem is over.
- Users and AP admins are often curious about, asked to, or want to credit the specific providers that have enabled their work and, in practice, need to summarize historical AP usage by site.

What’s Hard (5a/5)

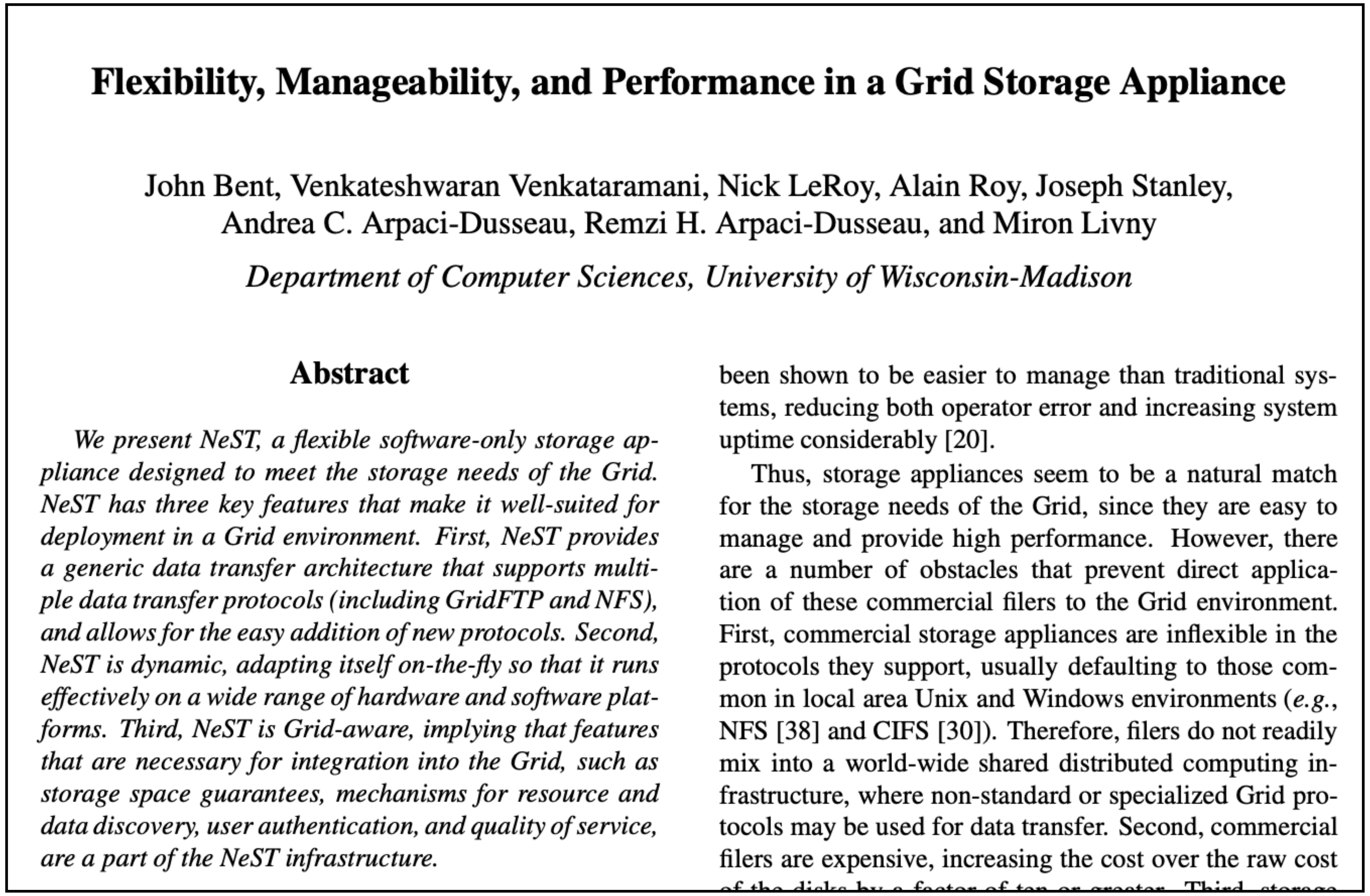
Data on the Grid!

- Kinds of data
  - job runtime data — running job inspection, debugging, progress monitoring
    - in a local pool, shared submit/execute filesystems and condor_ssh_to_job work well
    - neither work well on the grid (OS Pool, IGWN Grid, etc.)
    - makes no sense to always xfer/stream data back when you only need it once in a blue moon
    - this is a major practical usability regression for users moving from local pools to the grid (OS Pool, IGWN Grid, etc.)
  - job checkpoint data
    - used to be Condor’s problem to manage, now it’s each users’ problem — oh for the days of yore!
  - job/workflow input data (science data, parameters, configuration)
    - pushing everything from the submit node is easy for users but doesn’t scale…
    - CVMFS is easy for users but isn’t reliable…
  - intermediate workflow data (inter-job results)
    - back-and-forth to the submit node is state of the art for most LIGO-Virgo-KAGRA workflows
  - workflow output data (science output, workflow metadata, debugging info)
    - back to the submit node is state of the art for most LIGO-Virgo-KAGRA workflows
  - FUTURE: runtime telemetry (scientific and computing progress metadata)
    - should we do this /around/ Condor, in coordination with Condor (?), or via Condor (e.g., classads, chirp)?

What's Hard (5b/5)

Data on the Grid!

- **What are the right storage elements, and auth/xfer protocols for 3rd party xfer to move away from AP<->EP xfers?**
- I don't know yet…
- Exactly 20 years after the June 2002 Condor NeST paper, we still don't have DHTC storage discovery, authentication, authorization, allocation, QoS, timeout/cleanup, etc.
- But many promising building blocks are here (SciTokens, Stash, Rucio, etc.). Can we pull them together?



Flexibility, Manageability, and Performance in a Grid Storage Appliance

John Bent, Venkateshwaran Venkataramani, Nick LeRoy, Alain Roy, Joseph Stanley, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Miron Livny

Department of Computer Sciences, University of Wisconsin-Madison

Abstract

We present NeST, a flexible software-only storage appliance designed to meet the storage needs of the Grid. NeST has three key features that make it well-suited for deployment in a Grid environment. First, NeST provides a generic data transfer architecture that supports multiple data transfer protocols (including GridFTP and NFS), and allows for the easy addition of new protocols. Second, NeST is dynamic, adapting itself on-the-fly so that it runs effectively on a wide range of hardware and software platforms. Third, NeST is Grid-aware, implying that features that are necessary for integration into the Grid, such as storage space guarantees, mechanisms for resource and data discovery, user authentication, and quality of service, are a part of the NeST infrastructure.

been shown to be easier to manage than traditional systems, reducing both operator error and increasing system uptime considerably [20].

Thus, storage appliances seem to be a natural match for the storage needs of the Grid, since they are easy to manage and provide high performance. However, there are a number of obstacles that prevent direct application of these commercial filers to the Grid environment. First, commercial storage appliances are inflexible in the protocols they support, usually defaulting to those common in local area Unix and Windows environments (*e.g.*, NFS [38] and CIFS [30]). Therefore, filers do not readily mix into a world-wide shared distributed computing infrastructure, where non-standard or specialized Grid protocols may be used for data transfer. Second, commercial filers are expensive, increasing the cost over the raw cost

Discussion?