

Fast, Lightweight and Secure Cache-Invalidation with Pulse

Jacob Gorm Hansen <jacobg@diku.dk>

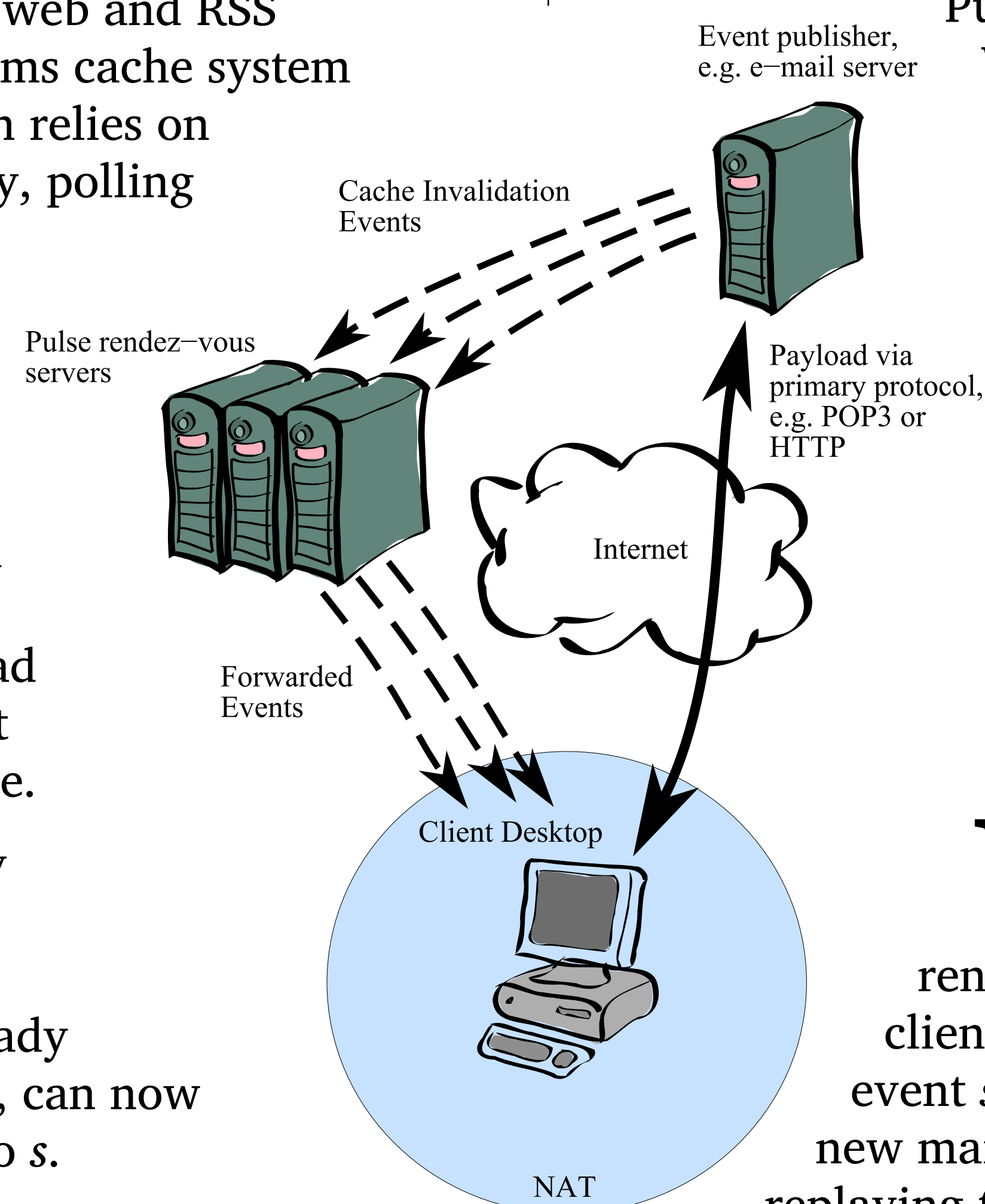
Callback Cache Invalidation

End-user computer systems such as laptops, desktops, handhelds and smartphones, increasingly cache replicas of remote information. Mail readers cache remote mailboxes, web and RSS readers cache news summaries, and operating systems cache system binaries. To stay synchronized, client software often relies on periodic polling of the master replica. Unfortunately, polling leads to a classic trade-off between server load/bandwidth use, and update latency. Distributed File Systems such as AFS have taught us that for read-mostly workloads, *callback invalidation* is often a better alternative.

Pulse is a system for callback cache-invalidation in the wide area. We assume the existence of a primary protocol, such as HTTPS or SSH, for payload propagation, and concern ourselves only with event notification, i.e. *when* but not *how* to get the update.

The central idea in Pulse is the use of a one-way function H for the verification of event authenticity. An invalidation event is published by releasing a secret s to a set of subscribers, who already know $H(s)$. Subscribers, and forwarders in the path, can now trivially verify event integrity by application of H to s .

Subscribers obtain $H(s)$ during synchronization, piggy-backed onto the primary protocol. Because s is random, the security and privacy guarantees of the primary protocol carry over to the cache-invalidation protocol. The decoupling of update notification and payload propagation makes Pulse protocol neutral and policy-free.



Real-time E-mail Notification

We have implemented Pulse as a prototype system. In practice, Pulse runs over DNS-formatted UDP packets, which means that Pulse traffic is able to traverse most NATs and firewalls. When a client is behind a NAT, it has to renew its event subscriptions at regular intervals, between 30 seconds and five minutes depending on the particular NAT configuration.

Real-time e-mail was our first Pulse application. Similar to a distributed file system, we use a log file to keep a local file system replica (the mail folder) in sync. The primary transport in the system is SSH. The procmail log file is used as the file system modification log. The client connects through SSH, replays the log file on its own replica, and receives a new Pulse event-id $H(s)$, where s is a new random secret kept by the mail server, to listen for.

When new mail arrives at the server, the procmail script sends a pulse event containing s to the rendez-vous servers, from where it gets forwarded to the client mail reader. The client verifies the integrity of an event s' , by checking if $H(s') = H(s)$. If the event is genuine, the new mail is fetched by initiating a new SSH connection, replaying the log file, and so forth.

Because s is a random number, and H is a cryptographically strong one-way function, the confidentiality and integrity properties of the primary SSH connection carry over to the Pulse event notification. With Pulse, e-mail is almost as fast as Instant Messaging.

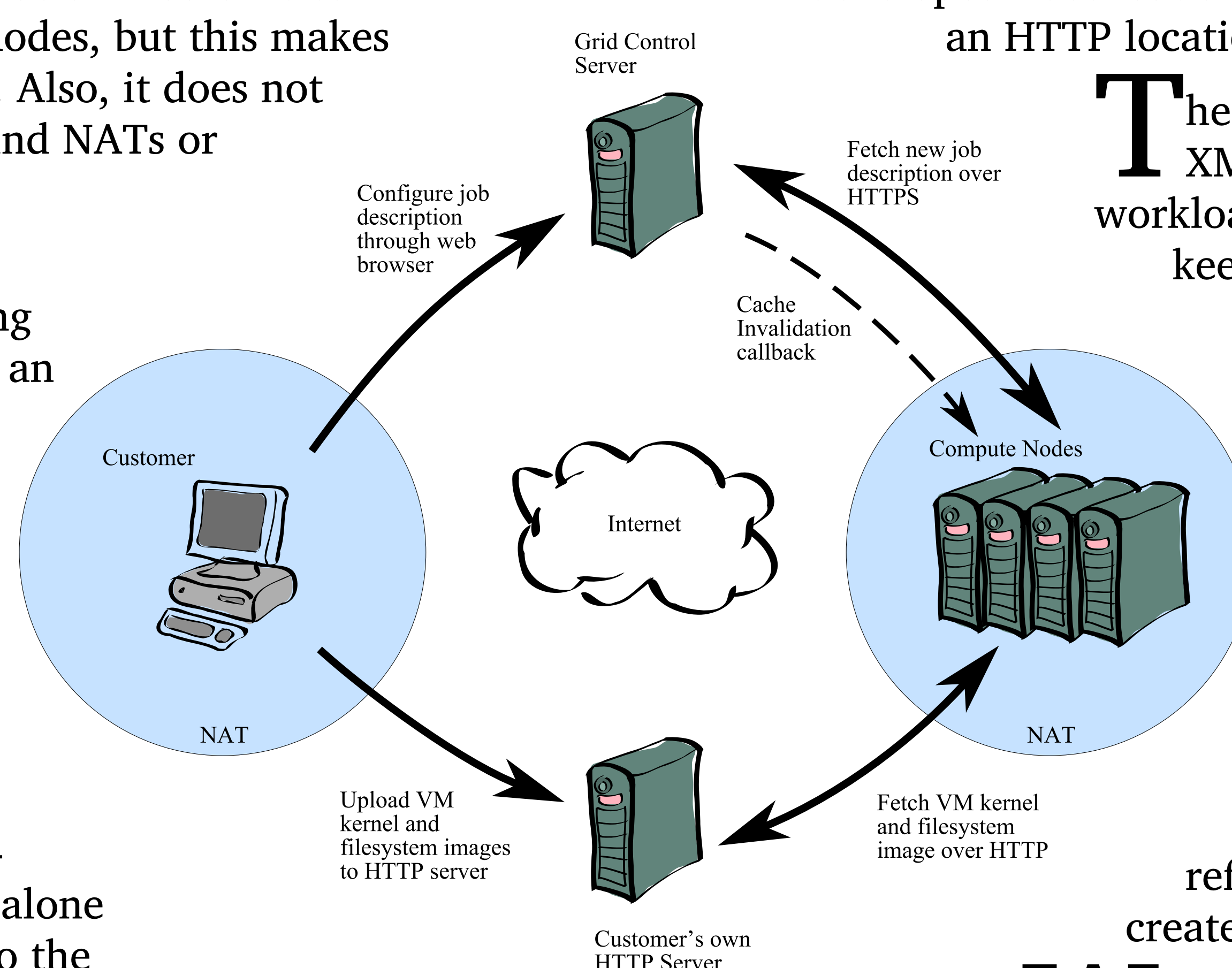
System Software Updates

We frequently have a need for updating the system software on a large number of compute nodes, but rarely have the patience to wait for them to periodically poll a central repository — especially not when dealing with security updates. One solution would be to network-mount the root file systems of all nodes, but this makes the root NFS server a single point of failure. Also, it does not work for those of our machines hidden behind NATs or firewalls.

Instead, we use Pulse to send out cache invalidation events to all the nodes, telling them to re-fetch their system software from an HTTP address. When not updating their caches, the machines function independently of the server. To save time, we delta-compress the updates using our EDelta executable delta compression algorithm.

EDelta runs in linear time and constant space, and efficiently encodes software updates. At the heart of EDelta is a Longest-Common-Substring (LCS) matcher, but LCS alone deals poorly with executable updates, due to the effects of code relocation. EDelta recognizes the effects of code-relocation, and encodes them efficiently.

With EDelta and Pulse, we can deploy updates to tens of megabyte of binaries, replicated at a large number of widely distributed nodes, in seconds.



On-Demand Compute System

Our ultimate application of Pulse is the management of a large and distributed on-demand computing system. Each node in the system hosts a number of Xen virtual machines. The VMs use specially developed boot-loader firmwares, with the ability to boot a VM off an HTTP location.

The workload description for each node is an XML file, cached from the master at the central workload management server. Pulse is used for keeping node workloads synchronized with the master.

The master contains a MySQL database controlling the workloads of all nodes. The database interfaces with an HTTP server that exposes a web user interface, and a web service that provides XML workload descriptions.

When the workload for a node is reconfigured, a Pulse callback invalidation is sent to the node. The node refreshes the workload, and modifies, creates or destroys VMs accordingly.

With this model, we are able to rapidly reconfigure a large number of nodes, even when the nodes are on different networks, or behind NATs and firewalls.

