Linked Data Platform for Web Applications

by

Joe Presbrey

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

June 2014

Copyright © 2014 Joe Presbrey.

The author hereby grants to M.I.T. permission to reproduce and distribute publicly
paper and electronic copies of this thesis and to grant others the right to do so.

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2014

Certified by .....
Tim Berners-Lee
Professor
Thesis Supervisor

Accepted by .....
Albert R. Meyer
Chairman, Masters of Engineering Thesis Committee

Linked Data Platform for Web Applications

by

Joe Presbrey

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2014, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Most of today's web applications are tightly coupled to proprietary server backends that store and control all user data. This thesis presents Linked Data as a decentralized web app platform, eliminating vendor lock-in, and separating user data from web apps giving users control over their data and where it's stored, independent of choice of application. Linked Data architecture replaces traditional app-data silos with a universal integration platform using global identifiers, shared ontologies, and a scalable, standardized data model. We provide 3 interoperable Linked Data server implementations in PHP, Python, and Go, and evaluate their performance. Traditional filesystems and relational databases have been integrated, and several new decentralized web apps have been developed for the platform. As of May 2014, worldwide open source community members are using and contributing to these compatible apps and servers, and the design continues to be refined and standardized at the W3C.

All code available at: `https://github.com/linkeddata`

Thesis Supervisor: Tim Berners-Lee
Title: Professor

Acknowledgments

I thank Tim Berners-Lee for his designs for this project and invaluable advice. I also thank Alexandre Bertails, Melvin Carvalho, Sandro Hawke, Anne Hunter, Lalana Kagal, Eric Prud'hommeaux, and Andrei Sambra for their collaboration and support. Finally, I thank the MIT SIPB and CSAIL TIG for providing open, scalable computing resources for our community.

Contents

Chapter 1

Introduction

Linked Data practices have already been widely successful in mass integration of bulk datasets. Wikipedia can be mined with semantic queries across any number of domains [3]. High volume commercial retailers embed metadata in product pages to enrich consumers search and shopping experience [1]. Several prominent governments are using Linked Data to improve the data integration workflows, by avoiding having to deal with huge diversity of APIs [11]. This also increases the transparency of the government and utility of the system.

In practice, its success is primarily related to the robustness principle [34]: be flexible in what you accept, strict in what you produce. In the context of Linked Data, the guidelines are simple enough to fit on a coffee mug (Figure 1-1). These practices facilitate optimistic interoperability at web scale: just as most web pages will render in any web browser, we want all data to be considered in any query. Web App developers are typically lenient with user input, but strict in production and expectation of their backend APIs and databases.

The directory of Web APIs maintained at ProgrammableWeb.com has grown 10x from 2009 to 2013 (Figure 1-2). Though LODStats reports 61 billion triples across 928 Linked Data datasets [2], no one has been able to count the data contained across the thousands of Web APIs. Integration of a single Web API into a Web App requires significant programmer involvement. Without basic compatibility insofar as to even count siloed data, its not likely that decisions based on a cohesive knowledge across so

Figure 1-1: Linked Data Best Practices

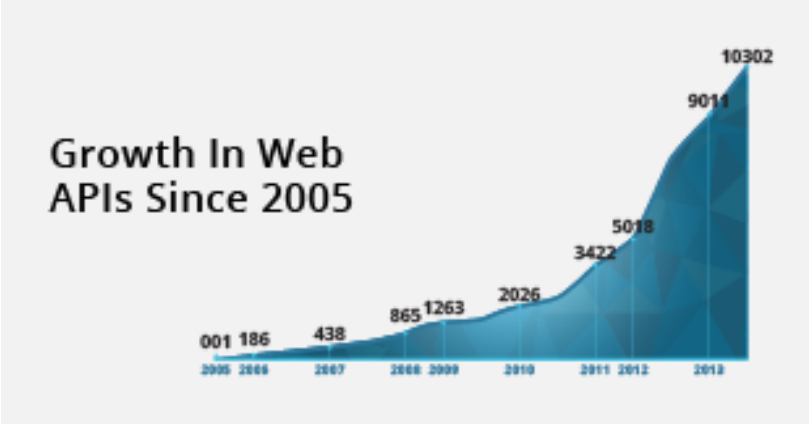

Figure 1-2: Web API growth - programmableweb.com

many disparate, proprietary sources will ever be possible. Therefore, an opportunity exists to improve this situation using a new architecture.

The inventor of the Web, Tim Berners-Lee (TimBL), has described the neccessary design elements over decades of web architecture development [5]. We refer to the composition of his designs as *the Linked Data Platform* (LDP). This architecture ends proliferation of proprietary Web APIs, by separating applications from data across a standardized HTTP API. Users of these applications wield more highly enriched and interlinked databases than that of any Web API, while retaining control of their own data, and freedom from vendor lock-in.

The contribution of this thesis is the creation and evaluation of three implementations of TimBL's designs. The rest of this paper is organized as follows: Chapter 2 presents technology and research background for Chapter 3, the design for the Linked Data Platform. Chapter 4 describes implementation considerations. Chapter 5 documents our benchmarks and tests of the systems. Chapter 6 describes web apps developed for the platform. Chapter 7 relates previous work, and Chapter 8 summarizes our conclusions and future work.

Chapter 2

Background

MIT has supported my excitement about developing web apps and data services in several ways over the past decade.

Shortly after I arrived at MIT as an undergraduate, I created `sql.mit.edu`, the first and foremost campus-wide MySQL service, hosting web app databases for over 5000 courses, faculty, students, and other groups, with support and hardware generously provided by the MIT Student Information Processing Group. The MIT SIPB SQL service primarily supports users of another service I co-created with MIT SIPB member Jeff Arnold, `scripts.mit.edu`. This service is open and free to the entire MIT community and is the leading web app hosting infrastructure on campus.

Next, MIT Information Services and Technology hired me to create a web-accessible gateway to AFS, MIT's officially supported networked filesystem. They chose the WebDAV protocol, described below in Section 2.1, the predominant protocol for remotely publishing and managing web documents at that time.

Finally, I met Professor Tim Berners-Lee (TimBL), inventor of the World Wide Web. His ideas and designs for web architecture immeasurably expanded my understanding of the practice and potential of web standards, apps, databases, and engineering. Section 2.2 below describes his foundations for the work of this thesis, *Linked Data*.

2.1 WebDAV

The *WebDAV protocol*[1] extends the *HTTP protocol*[2] to provide methods for collaborative content authoring among remote clients on the web. Many web content publishing services allow WebDAV for various uses such as online storage and backup like Apple's iDisk and collaborative calendaring like the CalDAV interface to Google Calendars. Its also common for a webmaster to use WebDAV to upload files and publish modifications to their website. Client support for the protocol is built into widely-deployed interfaces including Mac OS X Finder, Microsoft Web Folders, Adobe Dreamweaver, and many others. An Apache module called *mod_dav* implements server support for the protocol and provides the WebDAV filesystem store used in this project. Table 2.1 lists the methods handled by the module provided by the HTTP and WebDAV specifications respectively.

Specification	*Methods*
HTTP	OPTIONS, GET, PUT, DELETE
WebDAV	COPY, MOVE, MKCOL, LOCK, UNLOCK, PROPFIND, PROPPATCH

Table 2.1: HTTP Methods handled by *mod_dav*

For example, the PUT method can be used to upload a file by sending:

```
PUT /uploads/new_file.txt HTTP/1.1
```

to a WebDAV-enabled HTTP server with the file contents in the HTTP request body.

Data payload format is not part of the WebDAV standard. Understanding original document varieties, and formulating compatible patched documents, is burdensome on the clients and users. Additionally, minor changes to a single document require republishing the entire resource, and coarse-grained locking of the resource for concurrent updates.

[1] `http://www.ietf.org/rfc/rfc4918.txt`
[2] `http://www.ietf.org/rfc/rfc2616.txt`

```
{
  "@context": {
    "name": "http://xmlns.com/foaf/0.1/name",
    "homepage": {
      "@id": "http://xmlns.com/foaf/0.1/homepage",
      "@type": "@id"
    }
  },
  "homepage": "http://presbrey.mit.edu/",
  "name": "Joe Presbrey"
}
```

Figure 2-1: Example of JSON Linked Data (JSON-LD)

2.2 Linked Data

Linked Data is derived from the Semantic Web, an online graph of machine-readable data published in a shared document format, eg. JSON[3] (see Figure 2-1) or N3[4]. Documents expressing data in any number of shared vocabularies are published at universally resolvable, interlinked URIs producing *Linked Data* representing a shared, structured graph of knowledge. Prefixes are used to abbreviate common URI paths and can be dereferenced by substituting the associated URI. (Vocabularies used in this project are shown in Table 2.2). A query language called SPARQL[5] is used to query Linked Data across diverse sources and formats.

prefix	*URI*
acl:	`http://www.w3.org/ns/auth/acl#`
cert:	`http://www.w3.org/ns/auth/cert#`
rdf:	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
rdfs:	`http://www.w3.org/2000/01/rdf-schema#`
rsa:	`http://www.w3.org/ns/auth/rsa#`
stat:	`http://www.w3.org/ns/posix/stat#`

Table 2.2: Common Namespace Definitions

[3] `http://json-ld.org/`
[4] `http://www.w3.org/DesignIssues/Notation3`
[5] `http://www.w3.org/TR/rdf-sparql-query/`

Chapter 3

Design

This project's design is the result of decades of web architecture development by Tim Berners-Lee, the inventor of the World Wide Web [5]:

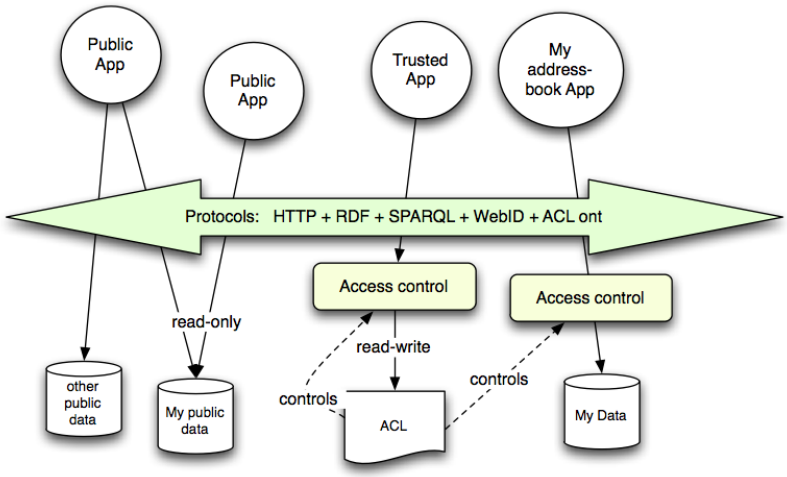

Figure 3-1: Linked Data Platform Design

The *Linked Data Platform* is a RESTful HTTP service like many other Web APIs today. But instead of arbitrarily structured JSON [8], all data exchanged between peers is modeled using W3C Linked Data standards as shown in Figure 3-1. RDF resources (documents and graphs) accumulate *triples* like relational database tables accumulate rows.

Most components are recommendations (or prospective) of W3C, or RFCs from IETF, composed with general web and unix principles. Since much of this project

is now part of active consensus processes, specifics of the platform such as protocols and formats may evolve after this thesis is published.

The platform is a client-server system, in which the servers are commodity HTTP servers, but enhanced using the features described in the following three sections: Querying, Updates, and Security. See also Tables 3.1, 3.2, and 3.3 for implementation comparisons.

3.1 Querying

Content Negotiation

Linked Data resources on the web do not need filename extensions seen in conventional filesystems. Clients tell the server how to respond with Linked Data resources using the standard HTTP `Accept` header [6], eg.

```
GET /joe/profile HTTP/1.1
Accept:  text/turtle
```

JSON, Turtle, and even XML are standard formats. The standard HTTP `Content-Type` header confirms formats returned from servers to clients, and allows clients to hint the format of submitted data. Consistent with the robustness principle, our servers will also guess if the format looks like an understood standard, eg. Turtle:

```
<joe#profile> foaf:name "Joe Presbrey".
```

Directory Listings

Web servers traditionally provide directory indexes for resources meant to be browsed or discovered, but these vary by server. LDP implementations facilitate discovery of resources by new integrators, and assist returning contributors by using Linked Data to generate listings for any collection (directory) of resources allowed for access by the agent. When backed by a posix-like filesystem, the data contain stat metadata from the filesystem using an ontology I contributed: `http://www.w3.org/ns/posix/stat`, eg.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix stat: <http://www.w3.org/ns/posix/stat#> .
@prefix ldp: <http://www.w3.org/ns/ldp#> .

<>  a ldp:BasicContainer, stat:Directory ;
    ldp:contains <sunset.jpg> .

<sunset.jpg>
    a stat:File ;
    stat:mtime "1398984524" ;
    stat:size "523020" .
```

Query String Filters

A developer often needs just one or a couple fields from a record. We make this easy by allowing query parameters to filter triples returned from a resource. For example,

```
GET //host/users/tim/profile?p=foaf:name,foaf:img HTTP/1.1
```

will return only names and photos in Tim's profile.

Resource Globbing

The utility of globbing is often demonstrated in command-line development. For example, a coarse search for images in group HTML profiles might look like:

```
grep "<img" groups/*/index.html
```

A similar but more exact, semantic query over Linked Data:

```
GET //host/groups/*/profile?p=foaf:img HTTP/1.1
```

Following the parameters of the URI in the request above, the server first accumulates graphs matching the URI globbing pattern, and then filters triples by the predicate specified, foaf:img in this case, finally returning eg.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
</groups/chess/profile> foaf:img <team.jpg> .
</groups/crew/profile> foaf:img <regatta2014.jpg> .
```

This functionality has not yet been addressed in standardization work.

SPARQL Queries

For more complex queries, we implement W3C's standardized SQL-like query interface for RDF: SPARQL [24]. Like all RDF on the server, SPARQL results are available in JSON, CSV, and other standard formats. Instead of SQL fields and values, SPARQL uses graph patterns to match RDF models. An example SPARQL query request follows:

```
POST /users/tim/profile HTTP/1.1
Content-Type: application/sparql-query
Accept: application/sparql-results+json
SELECT ?types { ?g rdf:type ?types }
```

3.2 Updates

LDP differs fundamentally from most existing HTTP based data services in that it provides efficient update of RDF resource data in real-time. The following features provide that functionality.

Entity Tags + If-Match

Originally designed for cache control [7], standard HTTP headers `E-Tag`, `If-Match`, `If-None-Match` are used during updates for concurrency control. When writing iterative updates to a resource, the previous `E-Tag` provided by a server is reissued with the client's next request to assure the resource has not had intermediate, possibly conflicting changes by other clients.

PUT, POST

Clients use a PUT request to create or overwrite a resource with the data given in the request body. The standard header `If-None-Match: *` is available to prevent clobbering. POST instructs the server to always append the data instead of create/overwrite.

PATCH

The HTTP PATCH [9] method partially updates a resource with new object

data for any subject+predicate triples mentioned in the request body. Existing objects for unmentioned subject+predicates remain unchanged. For example, the following request reflects a user updating their password:

```
PATCH /users/tim/profile HTTP/1.1
Content-Type: text/turtle
<#me> api:password "{sha1}3da541559918a808c2402bba5012f6c60b27661c" .
```

SPARQL Updates

When POST and PATCH are not enough, developers can use SPARQL Updates for more complex update semantics [25]. Common SQL-like publishing verbs include INSERT and DELETE. The following example demonstrates adding an arc in my FOAF social graph from me to Tim:

```
INSERT {
 <#me> foaf:knows <//www.w3.org/People/Berners-Lee/card#i>
}
```

Multiple queries may be joined in single request body (transaction) with the semi-colon.

WebSockets

HTML5 WebSockets [28] are used to support publishing and subscription of realtime graph updates, though experimental and nonstandard within the community. The update stream can be controlled by specifying a graph pattern at subscription time. Currently updates are replicated in SPARQL Update format, but an upcoming patch standardization is expected.

3.3 Security

Cross-Origin Resource Sharing

CORS is a critical feature for web app security [19]. The standard adds new HTTP headers which allow servers to control which foreign domains may send requests to local resources within a web browser. As part of the framework,

browsers send the `Origin` header to indicate the source domain of a web app making an HTTP request, eg.

```
Origin:  calendar.github.io
```

In my LDP implementations, this header is used in access control checks so users can control which apps may access their data using their browser credentials.

Federated Login

Web App developers generally want to maximize their reachable audience. We facilitate this by providing federated login capabilities from silo users, bridging the existing world of URI-based identities onto the platform. This includes users of Google Accounts, Yahoo, OpenID, and other URI-based identity schemes.

WebAccessControl

WAC is an ontology for controlling access to a given resource by a given agent [29]. LDP implementations use a standard HTTP header to indicate a document called an Access Control List (ACL) where rules should go for a given resource. For example rules for a user's profile could be indicated by this header in an HTTP response:

```
Link:  <profile,acl>, rel=acl
```

This feature was developed by TimBL, James Hollenbach, and I, and published during my time as an undergraduate researcher [12].

WebID Login

WebID is an evolving W3C standard providing universal, decentralized web accounts backed by Linked Data [27]. The current implementation dereferences the URI of the user's claimed ID and compares the key of a client's TLS certificate with the list of keys published in the user profile. If there is a match, the user is successfully authenticated. The protocol is novel, however the user experience when using and managing client SSL certificates in web browsers is widely criticized.

Chapter 4

Implementation

All code is free and available on the web at: `https://github.com/linkeddata`

I have developed several implementations of the Linked Data Platform. Its first incarnation was the Data Wiki [14] powered by Algae [15], W3C's Perl Module. At the time, read-only SPARQL was available on various servers at a single conventional endpoint: `//host/sparql`. The project made two important contributions: multiplexing read-write SPARQL across the URI space, and using the request URI as the base URI. But clients were generally written as curl command line scripts.

The following 3 server implementations included with this thesis extend this idea adding support for decentralized web apps using the latest standards and recommendations from W3C and RFCs from IETF as described in Chapter 2. The software currently requires Redland RDF Libraries [4] and has been tested on Linux, OSX, and Windows.

I started with a PHP implementation due to the wide predominance of the LAMP web server stack (Linux-Apache-MySQL-Perl/PHP/Python) [1]. According to Netcraft as of January 2013, PHP runs nearly 250M sites and on 2.1M of 4.3M web servers [16]. So after a successful proof of concept in Perl, PHP was a natural target in supporting a large number, almost majority of servers. The following sections further describe ldphp and the evolution to the Python and Go implementations.

[1] `http://en.wikipedia.org/wiki/LAMP_(software_bundle)`

Table 4.1: Feature Comparison across implementations

Functionality:	ldphp	ldpy	gold
Content Negotiation [6]	●	●	●
Cross-Origin Resource Sharing [19]	●	○	●
Entity Tags + If-Match [7]	●	○	●
Federated Login: Google, OpenID, ...	●	○	○
JSON-LD [18]	●	○	●
PATCH [9]	●	○	●
Query String Filters	●	○	○
RDF Directory Listings	●	●	●
RDF Resources	●	●	●
Resource Globbing [32]	●	○	●
SPARQL Queries [24]	●	●	●
SPARQL Updates [25]	●	○	●
SPARQL Update WebSockets	●	○	○
WebAccessControl [29]	●	○	●
WebID Login [27]	●	○	●

Table 4.2: Resource Type Comparison for Queries

`{GET,HEAD} Accept:`	ldphp	ldpy	gold
application/ld+json [18]	●	○	●
application/n-triples [20]	●	●	●
application/rdf+json [21]	●	●	●
application/rdf+xml [22]	●	●	●
application/sparql-results+json [23]	●	●	●
text/turtle [26]	●	●	●
HTML, CSS, JS, images	●	○	●
Other octet streams	○	○	●

Table 4.3: Resource Type Comparison for Updates

`{PATCH,POST,PUT} Content-Type:`	ldphp	ldpy	gold
application/ld+json [18]	●	○	●
application/n-triples [20]	●	●	●
application/rdf+json [21]	●	●	●
application/rdf+xml [22]	●	●	●
application/sparql-update [25]	●	●	●
text/turtle [26]	●	●	●

4.1 ldphp

ldphp runs using mod_php embedded in Apache or using FastCGI mode of any web server. I have personally hosted ldphp LDP sites on Apache2+mod_fcgid (`scripts.mit.edu`), lighttpd, and nginx web servers. Due to the wide availability and longevity of PHP, this is oldest implementation, has received the most community contributions, and has the most expansive featureset.

ldphp works well for basic requests, but has more overhead than gold for PUTs and requests where more complex queries requiring concurrent network subrequests and/or aggregation of member resources. Performance evaluation is described further in Chapter 5.

4.2 ldpy

ldpy is a Linked Data Platform implementation for Python, another popular scripting language. Thanks to Python and Flask's modular superiority over PHP, this implementation is easy to compose with other platforms, despite the limited featureset. For example, it was used in the MIT CSAIL Big Data project to open a public MIT Warehouse dataset [2]. This implementation is backed by a commercial Oracle database yet still provides data interoperability within our platform.

This implementation leverages the Flask microframework to provide RESTful request dispatching for our platform. It leverages WSGI (Web Server Gateway Interface) with 100% compliance, a standard interface between web servers and Python that promotes web application portability across web servers [17]. The emergence of multicore computing has ultimately led us away from Python due to its limitations in high concurrency environments. As a result, ldpy is a minimal implementation of our platform in only 244 lines – see Table 4.1 for a feature comparison.

[2] `http://bigdata.scripts.mit.edu/warehouse/`

4.3 gold

gold is a Linked Data Platform implementation for Google's recently open-sourced language Go, a concurrent, garbage-collected language with fast compilation. By its design, Go proposes an approach for the construction of system software on multicore machines [10]. It provides a simple API to C libraries (librdf) and a native TLS web server, among other strong core libraries, and compiles into a single distributable binary, easy to install on Linux, OSX, Windows, x86, x86-64, and ARM, among others. The project currently includes 70 unit tests with nearly 500 assertions covering about 90% of the codebase.

Chapter 5

Performance

This chapter evaluates the implementations described in Chapter 4 side-by-side on a Linux server running Fedora 20, kernel 3.14.4-200.fc20.x86_64. The host has 8 cores of Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz and 16GB of RAM. We run 8 processes of each implementation to ensure saturation, with 10 trials of 100,000 requests (samples) each requiring several hours to complete in total. All configuration needed to reproduce these results is included in the Appendix.

5.1 Methodology

We seek benchmarks for Queries and Updates to supplement independent results given for the Security components [12]. Since low concurrency, bulk Linked Data solutions have already been proven, we measure transaction rate for small workloads with high concurrency using boom[1], a HTTP(S) load generator (ApacheBench replacement) written in Go.

The GET benchmark in Figure 5-1 repeatedly requests Turtle for the resource /. The server has no index.html so it creates a listing for the root directory on the fly in Linked Data. In this case, the directory contains three resources, each described by: type, ctime, mtime, and size. The PUT benchmark in Figure 5-2 repeatedly submits a single triple (`<a> <b> <c>.`) to resource /abc.

[1] `https://github.com/rakyll/boom`

Table 5.1: Benchmark Harness Comparison

	ldphp	ldpy	gold
direct HTTP		•	•
direct TLS			•
FastCGI via nginx	•		•
uwsgi via nginx		•	

Table 5.2: Benchmark Harness Versions

golang	1.2.2
gunicorn	18.0
nginx	1.6.0
openssl	1.0.1e
php	5.5.12
python	2.7.5
raptor	2.0.9
redland	1.0.16
uwsgi	1.9.19

We are interested in how many requests in a given interval can be completed successfully. Success means that the method had the desired result and the server kept functioning without any observed problems while also handling requests from a number of other concurrent clients.

We compare our implementations using common, modern web server harnesses. PHP is served using FastCGI via nginx, with Zend OPcache which provides faster execution through opcode caching and optimization. Two Python harnesses provide HTTP: gunicorn serving natively, and uwsgi via nginx. Three Golang harnesses are compared: built-in HTTP, TLS, and FastCGI servers.

5.2 Results

For all concurrency levels, gold server (Go) produces leading measurements for both tests, shown in Figures 5-1 and 5-2. Measuring against a NOOP server suggests only 3..4x overhead for a Linked Data Platform GET vs. a request doing absolutely

nothing, shown in Figure 5-3. PUT overhead is slightly higher.

Distributed Computing literature benchmarks frequently measure hash table performance using a workload of: 90% reads, 9% adds, 1% deletes. If we align reads with GET, and adds with PUT, our measurements of PUTs at 25..30% slower than GETs are superior in the expected workload.

Though the leading Go implementation is already highly performance, we further consider potential optimizations using the Go CPU Profiling tool, pprof, in Figure 5-4. Nearly 67% of CPU time is spend in our C RDF library, redland, 8% is spent in the CGo bridge, and 11% are spent in OS syscalls. This leaves only 14% remaining CPU time to consider in future optimizations to our Go code.

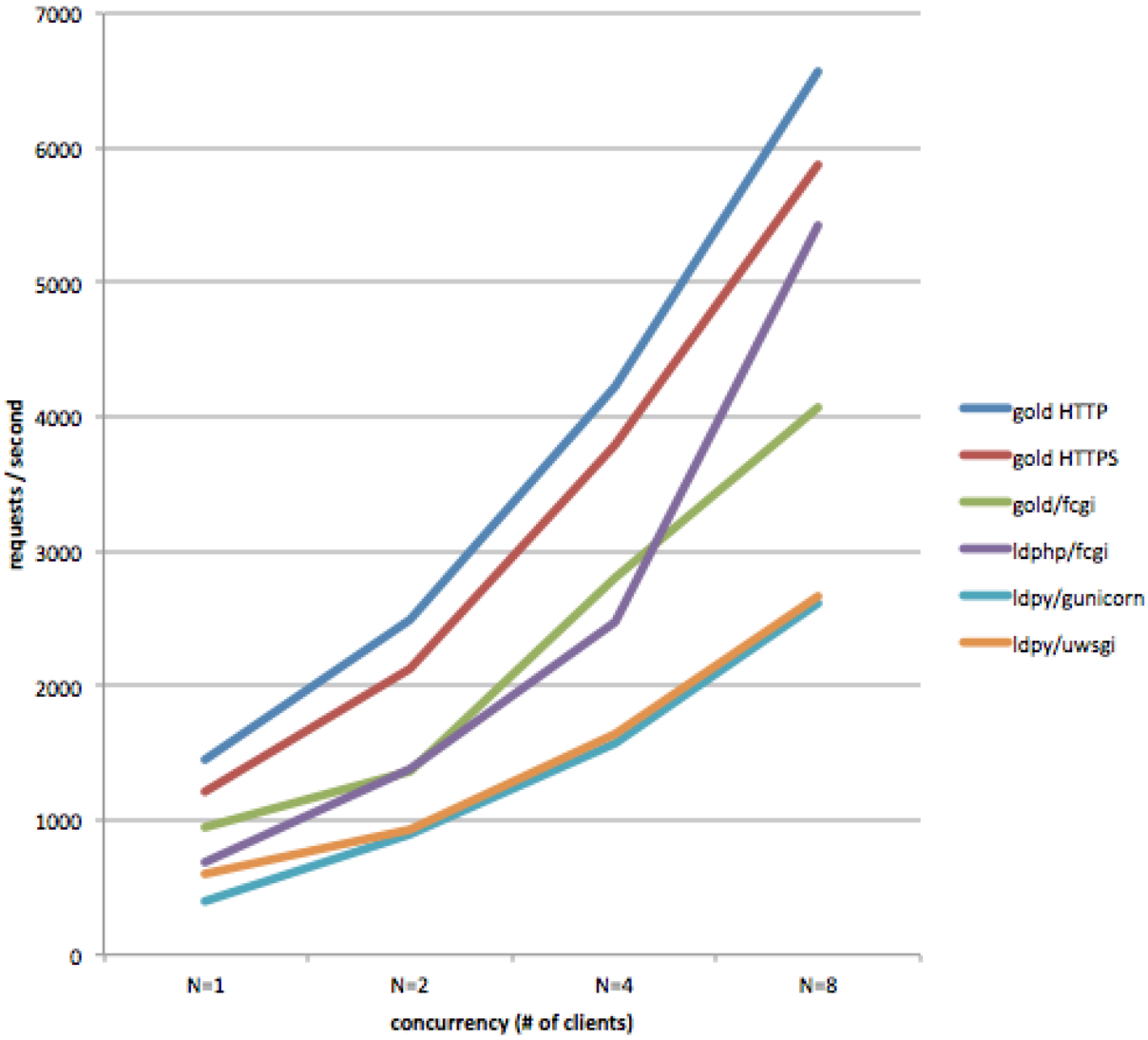

Figure 5-1: GET Benchmark Results

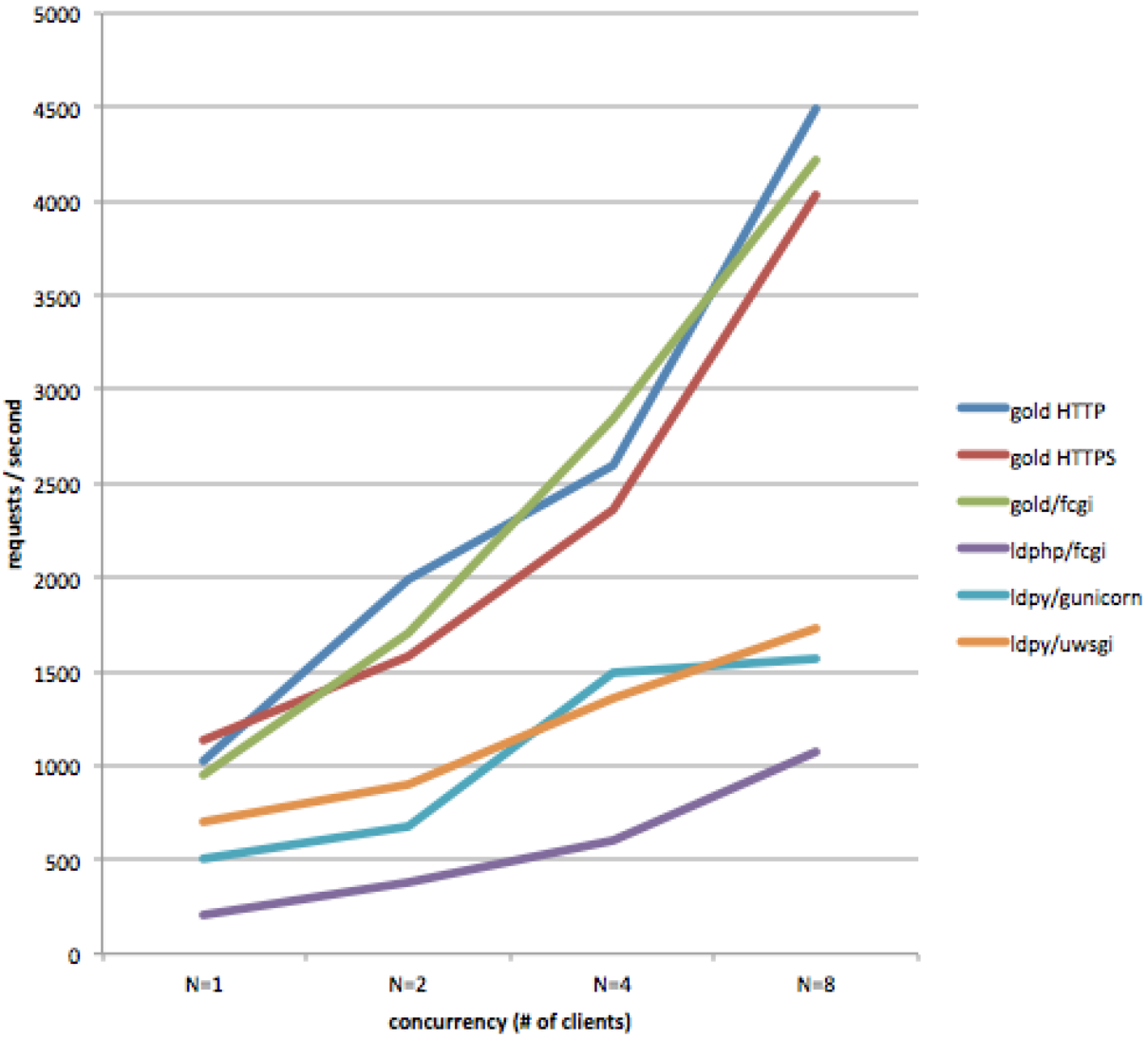

Figure 5-2: PUT Benchmark Results

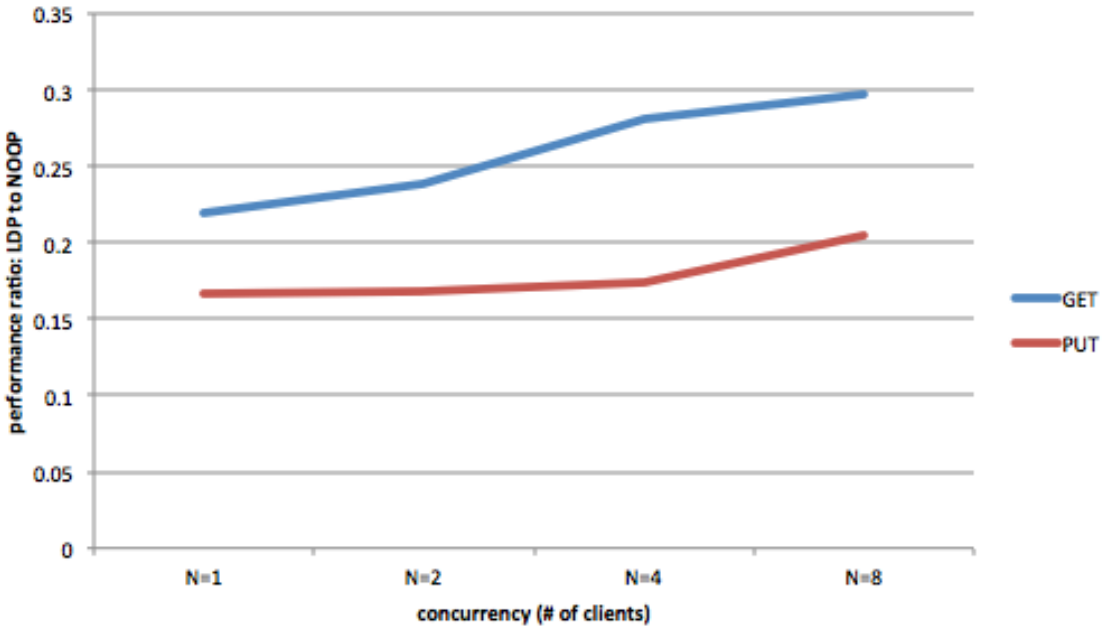

Figure 5-3: Overhead of gold LDP vs. NOOP

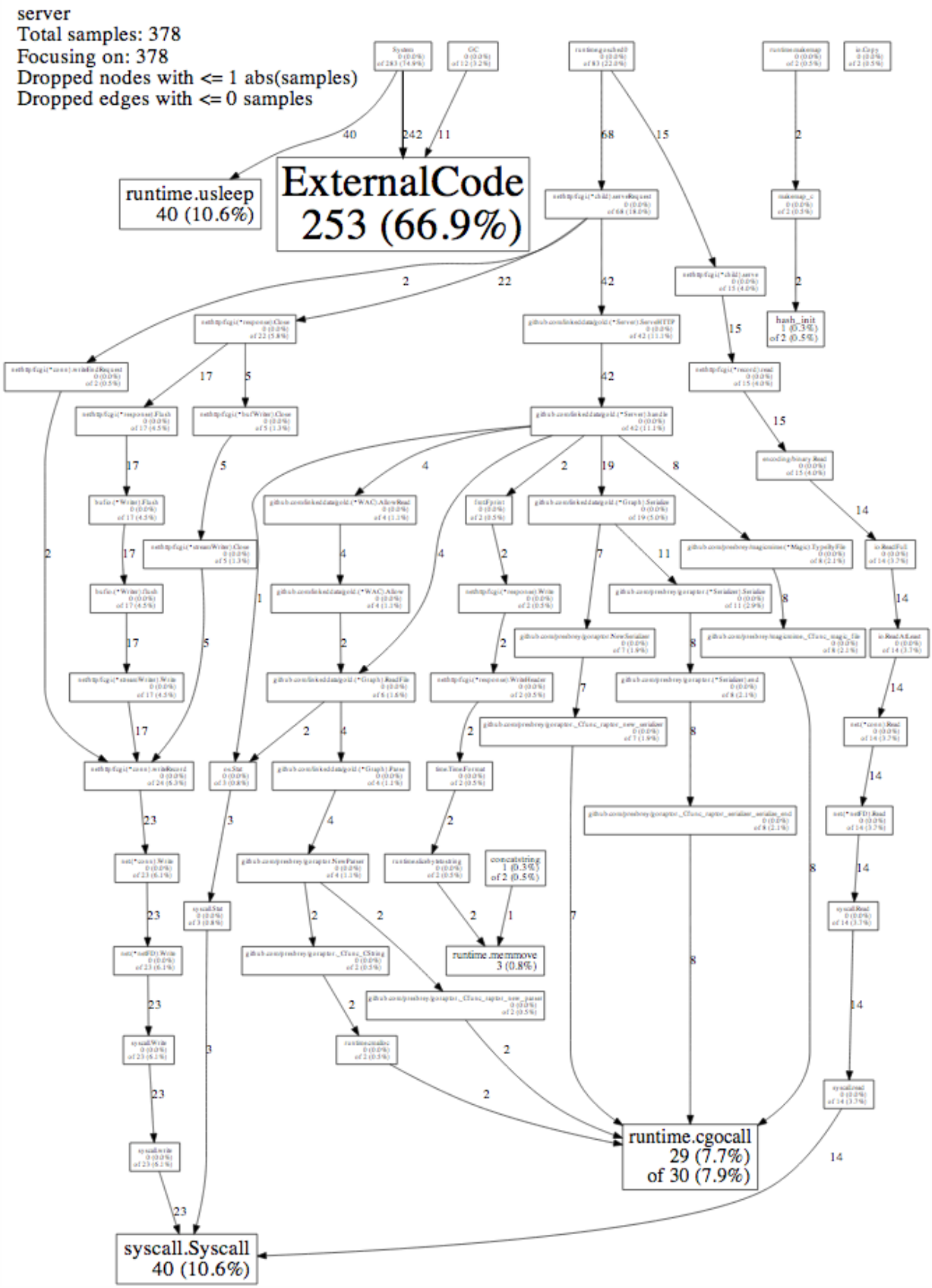

Figure 5-4: pprof CPU Profile: gold

Chapter 6

Applications

Linked Data Web Apps are generally implemented exclusively in client-side JavaScript. A key benefit is that anyone creative can make an app and sell it, as new apps can create and control access to data resources in the existing storage cloud.

The sections below describe Linked Data Web Apps, libraries, and extensions developed for the platform by Tim Berners-Lee (TimBL), our research group, others in our community, and me.

6.1 rdflib.js

rdflib.js is an RDF-based library developed by TimBL, with major contributions from his students in the MIT CSAIL Decentralized Information Group, implementing the protocols, codecs, and tools necessary for providing Linked Data robustness in a web browser or other JavaScript environment. I contributed several improvements including: adding read-write support for LDP using WebDAV and SPARQL, adding pubsub support using WebSockets, adding jQuery AJAX integration, and nodeunit tests. The library is available at:

`https://github.com/linkeddata/rdflib.js`

Figure 6-1: Tabulator: Linked Data Browser/Editor

6.2 Tabulator

Tabulator is a generic data browser, powered by rdflib.js, also created by TimBL and the DIG group. Its primary outline view shown in Figure 6-1 provides tree-based traversal over Linked Data graphs. Many rule-based views/tabs reconfigure automatically when node data is dereferenced based on rdf:type and other node properties.

The project started as a Firefox Extension. I improved performance, integrated new rdflib.js features, and ported it to the Chrome web browser. The source code is available at:

`https://github.com/linkeddata/tabulator`

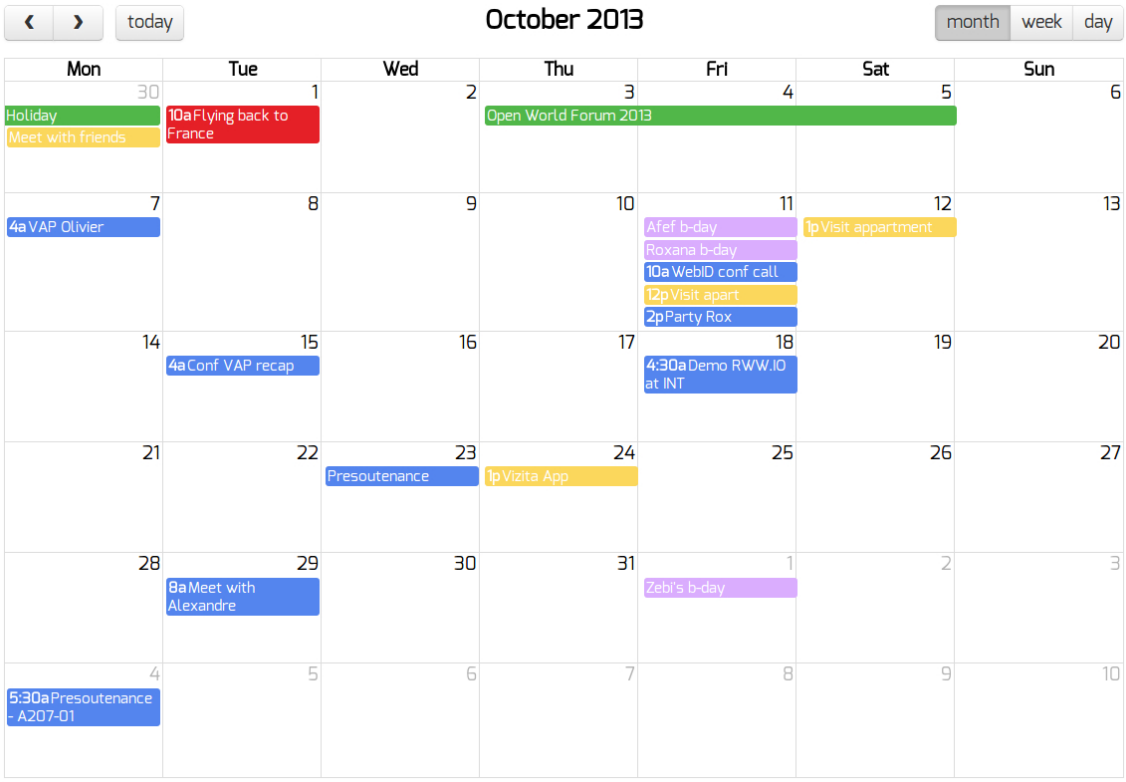

Figure 6-2: rww-apps.github.io/ld-cal

The browser extension is available at:

`https://chrome.google.com/webstore/detail/tabulator/lcfhlnpimhljekhgdohcldfmehlkfoae`

6.3 Calendar

Andrei Sambra developed a decentralized calendar web app for LDP shown in Figure 6-2. The app runs in a web browser from our shared community space on github, while his users' data remains private on their personal LDP server. Andrei currently runs ldphp, described in Section 4.1, on his personal LDP server, but his app is compatible with any implementation. The data can also be viewed in Tabulator and other blogging apps. The code for his web app is available at:

`https://github.com/rww-apps/ld-cal`

6.4 Microblog

Andrei Sambra also developed a blogging web app for LDP, shown in 6-3. Similar to the calendar app, CIMBA runs in a web browser from our shared community space on github, while each participant's data remains in control of their personal LDP server. Andrei made contributions to ldphp and gold as part of this work.

6.5 Spreadsheet

I created LD spreadsheet, shown in Figure 6-4 running from github. Any LDP server can be used to host the data. In developing the app, I contributed open/save dialog box widgets compatible with other LDP web apps shown in Figure 6-5.

6.6 Voting

Melvin Carvalho developed a voting web app for LDP shown in Figure 6-6. He ran several community votes including one inquiring whether Linked Data sites should participate in the SOPA blackout. Melvin currently runs ldphp on his personal LDP server.

6.7 mod_authn_webid

I developed this C module for the Apache2 web server to provide WebID authentication interoperability into the traditional Apache/.htaccess authentication pipeline. The project has a measured overhead of only a couple milliseconds [1], and has received contributions from Caltech. The code is available at:

`https://github.com/linkeddata/mod_authn_webid`

[1] `http://dig.csail.mit.edu/2009/presbrey/UAP.pdf`

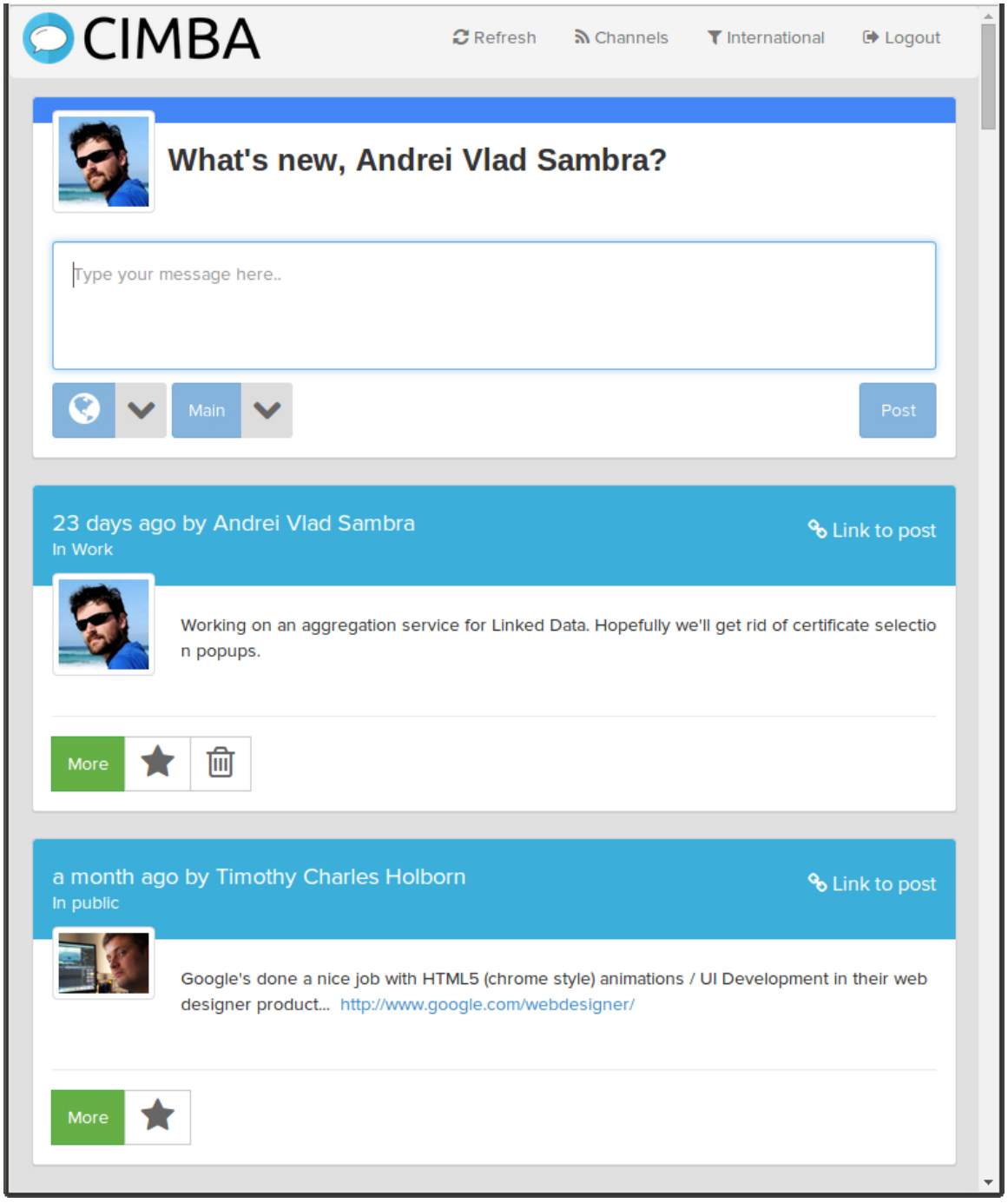

Figure 6-3: cimba.co

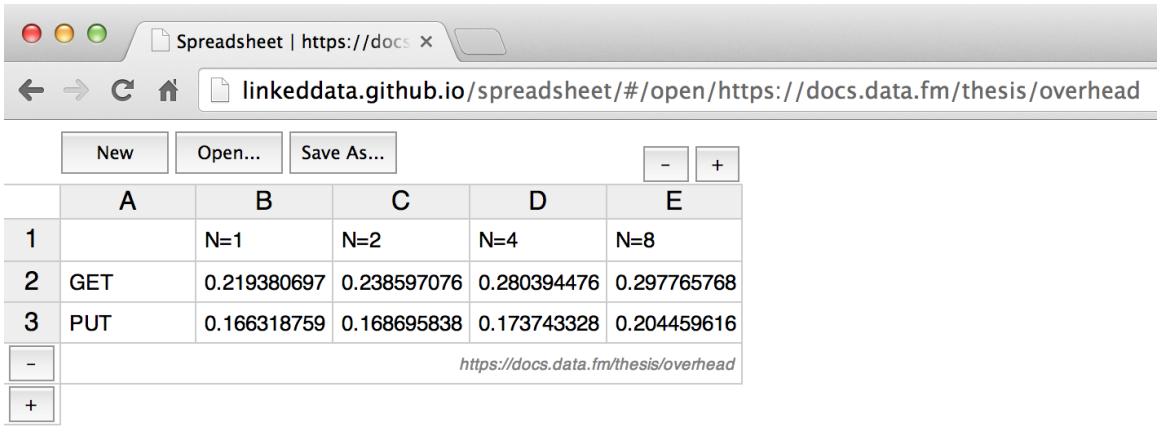

Figure 6-4: linkeddata.github.io/spreadsheet

Figure 6-5: Open/Save dialog

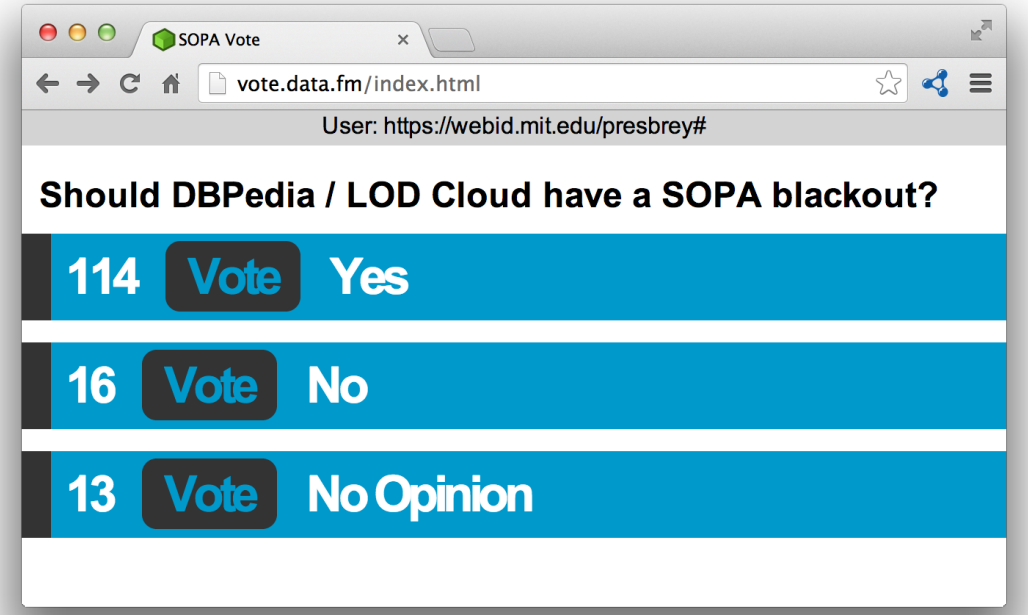

Figure 6-6: vote.data.fm

6.8 WebID.MIT.EDU

I created the MIT WebID service shown in Figure 6-7 to help bootstrap a web of trust using Linked Data profiles, that allocates identifers to MIT community members to match their MIT Athena identities. Later I added support for Federated Login with Google Accounts, though the web app is fundamentally compatible with any web authentication scheme.

Its unique contribution is providing life-cycle management of multiple crypto keys across multiple devices, and indirection to additional Linked Data profiles. The code is available at:

`https://github.com/linkeddata/webid.mit.edu`

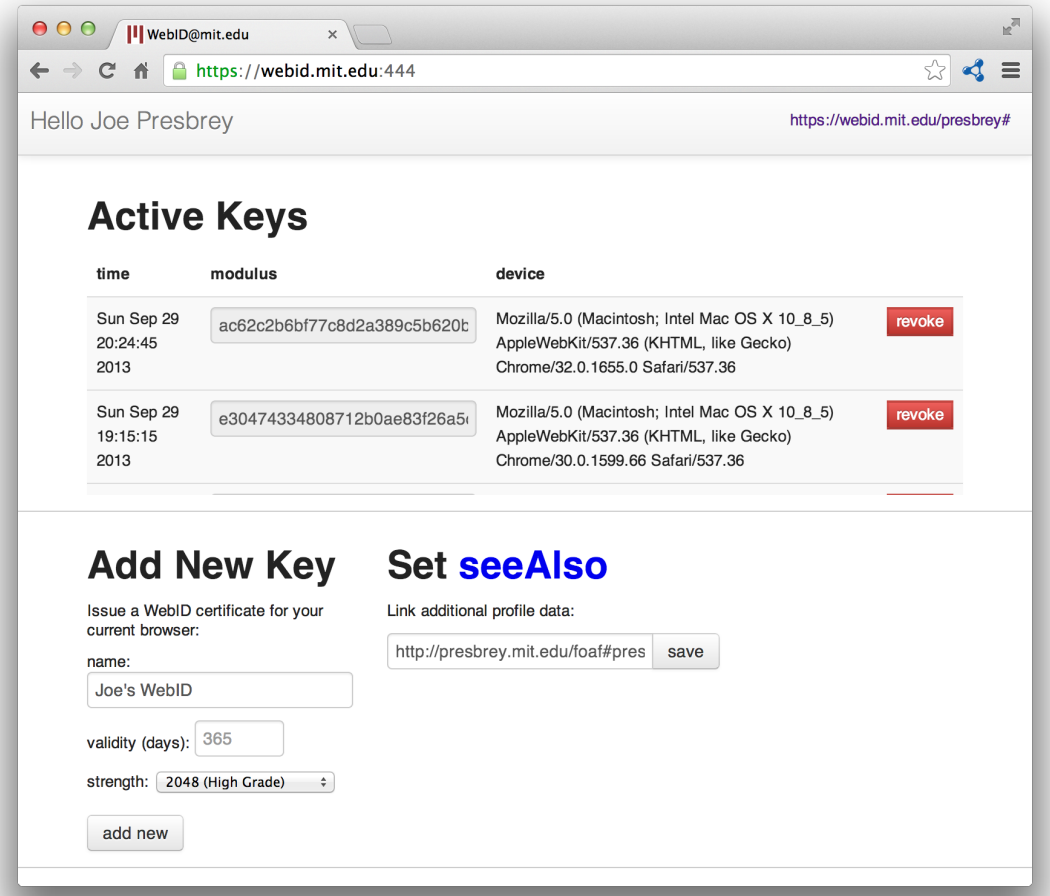

Figure 6-7: WebID.MIT.EDU

Chapter 7

Related Work

There are two tracks of work related to Linked Data Platforms for Web Applications: commercial API Management products, and the `unhosted.org` community. We describe our relationship to these in turn.

7.1 API Management

API management is used in the cloud to manage APIs that companies publish for use by mobile and Web business-to-consumer or business-to-business projects, with applications frequently developed by their business partners. For example, Appcelerator[1] describes itself as built to access heterogeneous data sources via a modern cloud architecture. Listed benefits include: eliminating the complexity of managing multiple APIs, lowering the cost of development and maintenance; and orchestration of data from heterogeneous sources, both public and enterprise, for richer, more transformative app experiences.

Similarly, Intel acquired Mashery.com in May 2013 and advertises their integrated offering as Composite API Management [2]. Data Format mediation is provided, with support for conversion of unstructured, semi-structured and structured XML data into RESTful API responses.

[1] `http://www.appcelerator.com/platform/apis/`
[2] `https://cloudsecurity.intel.com/solution-briefs/composite-api-management`

Gartner provides further analysis[3] of API governance and management products. Though these products may encourage local optimizations in API integration, no API management or consolidation vendor has advertised products with as broad ambitions as Semantic Web and Linked Data technologies.

7.2 unhosted.org

The community at unhosted.org shares many common ideals with Linked Data Platform practitioners. Their first reaction to disparate Web API growth was to build new web apps without any backends at all, also known as "serverless", "client-side", or "static" web apps. Unhosted app publishers do not see any user data since data does not go to servers. There are no servers – all storage is local, making running the apps inexpensive, highly decentralized, and scalable. However, user data is not interoperable between apps and across networks and organizations.

remoteStorage[4] is an addon for unhosted.org apps which provides decentralized user storage for unhosted apps using a common API. Developers behind remoteStorage have published a standard for the protocol with the IETF [5]. The fundamentals of their data protocol do not share the same standards underpinning Linked Data Platforms such as using URIs as global identifiers. Data schemas are defined in each application, possibly limiting interoperability between web apps. However, their decentralized, private storage work provides the same abstract benefits to users.

[3] http://www.gartner.com/technology/reprints.do?id=1-1IBXGIT&ct=130809&st=sb
[4] http://remotestorage.io/
[5] http://tools.ietf.org/html/draft-dejong-remotestorage-01

Chapter 8

Conclusions and Future Work

This thesis provides a solid base for future projects implementing collaborative authoring and publishing of Linked Data, with Web Access Control, for Web Applications. It provides three implementations of RDF-based decentralized application platforms. Query and update performance are comparable with traditional Web APIs. Users and developers can quickly and easily integrate new data sources without having to add new code to their applications. A user could, at the present time, use any implementation provided to escape vendor lock-ins and data silos, which is a notable advance over the recent progress of web centralization and monopoly.

The three server implementations demonstrate it is possible to streamline this technology in any web server, provide custom yet interoperable endpoints integrated with existing relational database technologies, and maximize concurrency and performance in multicore environments. Wide adoption will make life easier for users while maintaining a fully decentralized architecture in which identities, data storage, and applications can all be independent and managed by different sites.

Several client web apps such as a Calendar, Microblog, and Spreadsheet have been developed for the platform. By plugging services such as policy-aware query processors and data access logs into our system, one could create a complete policy-aware system based in an entirely decentralized environment, realizing the full potential of the Semantic Web. As of May 2014, groups at the W3C have adopted most components of this work and continue developing, refining, and standardizing the platform.

Bibliography

[1] Ashraf, J., Cyganiak, R., O'Riain, S., & Hadzic, M. (2011, March). Open eBusiness Ontology Usage: Investigating Community Implementation of GoodRelations. In LDOW. URL: http://ceur-ws.org/Vol-813/ldow2011-paper12.pdf.

[2] AKSW. LODStats. May 2014. URL: http://stats.lod2.eu/.

[3] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. In The semantic web (pp. 722-735). Springer Berlin Heidelberg. URL: http://w.websemanticsjournal.org/index.php/ps/article/viewFile/164/162.

[4] D. Beckett. Redland RDF Libraries. March 2014. URL: http://librdf.org/.

[5] T. Berners-Lee. Design Issues. May 2014. URL: http://www.w3.org/DesignIssues/.

[6] T. Berners-Lee; R. Fielding; L. Masinter. Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616): Content Negotiation. June 1999. URL: http://www.w3.org/Protocols/rfc2616/rfc2616-sec12.html.

[7] T. Berners-Lee; R. Fielding; L. Masinter. Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616): Entity Tags. June 1999. URL: http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.11.

[8] Crockford, Douglas. JavaScript Object Notation (JSON). 2006. URL: http://tools.ietf.org/html/rfc4627.

[9] L. Dusseault; J. Snell. PATCH Method for HTTP (RFC 5789). March 2010. URL: http://tools.ietf.org/html/rfc5789.

[10] The Go Authors. Frequently Asked Questions. April 2014. URL: http://golang.org/doc/faq.

[11] Hendler, James and Holm, Jeanne and Musialek, Chris and Thomas, George. US government linked open data. IEEE Intelligent Systems. 2012. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6185527.

[12] J. Hollenbach, J. Presbrey, T. Berners-Lee. Using RDF Metadata To Enable Access Control on the Social Semantic Web. ISWC 2009. URL: http://dig.csail.mit.edu/2009/Papers/ISWC/rdf-access-control/paper.pdf.

[13] L. Lamport. Paxos Made Simple. 2001. URL: http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf.

[14] J. Presbrey. Data Wiki. 2007. URL: http://dig.csail.mit.edu/2007/wiki/.

[15] E. Prud'hommeaux. Algae2 Perl Module. 1999. URL: http://www.w3.org/1999/02/26-modules/User/Algae-HOWTO.html.

[16] Netcraft. PHP just grows & grows. January 2013. URL: http://news.netcraft.com/archives/2013/01/31/php-just-grows-grows.html.

[17] P. Eby. PEP 333 – Python Web Server Gateway Interface. March 2011. URL: http://legacy.python.org/dev/peps/pep-0333/.

[18] W3C. A JSON-based Serialization for Linked Data. January 2014. URL: http://www.w3.org/TR/json-ld/.

[19] W3C. Cross-Origin Resource Sharing. January 2014. URL: http://www.w3.org/TR/cors/.

[20] W3C. RDF 1.1 N-Triples. February 2014. URL: http://http://www.w3.org/TR/n-triples/.

[21] W3C. RDF 1.1 JSON Alternate Serialization (RDF/JSON). November 2013. URL: http://www.w3.org/TR/rdf-json/.

[22] W3C. RDF 1.1 XML Syntax. February 2014. URL: http://http://www.w3.org/TR/rdf-syntax-grammar/.

[23] W3C. Serializing SPARQL Query Results in JSON. June 2007. URL: http://http://www.w3.org/TR/rdf-sparql-json-res/.

[24] W3C. SPARQL Query Language for RDF. January 2008. URL: http://www.w3.org/TR/rdf-sparql-query/.

[25] W3C. SPARQL 1.1 Update. March 2013. URL: http://www.w3.org/TR/sparql11-update/.

[26] W3C. RDF 1.1 Turtle. February 2014. URL: http://http://www.w3.org/TR/turtle/.

[27] W3C. WebID 1.0. March 2014. URL: http://www.w3.org/2005/Incubator/webid/spec/identity/.

[28] W3C. The WebSocket API. September 2012. URL: http://www.w3.org/TR/websockets/.

[29] W3C Community Wiki. WebAccessControl. March 2014. URL: http://www.w3.org/wiki/WebAccessControl.

[30] What is Linked Data?. data.gov.uk. January 2014. URL: http://data.gov.uk/linked-data.

[31] Wikipedia. Andrew File System. May 2014. URL: http://en.wikipedia.org/wiki/Andrew_File_System.

[32] Wikipedia. glob (programming). May 2014. URL: http://en.wikipedia.org/wiki/Glob_(programming).

[33] Wikipedia. JSONP. May 2014. URL: http://en.wikipedia.org/wiki/JSONP.

[34] Wikipedia. Robustness Principle. May 2014. URL: http://en.wikipedia.org/wiki/Robustness_Principle.

Chapter 9

Appendix

Benchmark Config: nginx.conf

```
daemon off;
error_log stderr;
pid /tmp/nginx.pid;
user nginx nginx;
worker_processes 2;

events {
    multi_accept on;
    use epoll;
    worker_connections 4000;
}

http {
    access_log off;
    client_body_timeout 10;
    index index.php index.html index.htm;
    keepalive_timeout 1;
    sendfile on;
    reset_timedout_connection on;
    send_timeout 2;
    tcp_nodelay on;
    tcp_nopush on;

    proxy_http_version 1.1;
    proxy_set_header Connection "";
    proxy_set_header Host $host;
```

```
upstream gold {
    server unix:/tmp/gold1.sock;
    server unix:/tmp/gold2.sock;
    server unix:/tmp/gold3.sock;
    server unix:/tmp/gold4.sock;
    server unix:/tmp/gold5.sock;
    server unix:/tmp/gold6.sock;
    server unix:/tmp/gold7.sock;
    server unix:/tmp/gold8.sock;
    keepalive 128;
}

upstream ldpy {
    server unix:/tmp/ldpy1.sock;
    server unix:/tmp/ldpy2.sock;
    server unix:/tmp/ldpy3.sock;
    server unix:/tmp/ldpy4.sock;
    server unix:/tmp/ldpy5.sock;
    server unix:/tmp/ldpy6.sock;
    server unix:/tmp/ldpy7.sock;
    server unix:/tmp/ldpy8.sock;
    keepalive 16;
}

upstream php {
    server unix:/tmp/php1.sock;
    server unix:/tmp/php2.sock;
    server unix:/tmp/php3.sock;
    server unix:/tmp/php4.sock;
    server unix:/tmp/php5.sock;
    server unix:/tmp/php6.sock;
    server unix:/tmp/php7.sock;
    server unix:/tmp/php8.sock;
    keepalive 16;
}

server {
    server_name goldfcgi;
    listen 8001 default;
    fastcgi_keep_conn on;
    location / {
        include /etc/nginx/fastcgi_params;
        fastcgi_pass gold;
    }
```

```
    }

    server {
        server_name ldpywsgi;
        listen 8003 default;
        location / {
            include /etc/nginx/uwsgi_params;
            uwsgi_pass ldpy;
        }
    }

    server {
        server_name ldphp;
        listen 8004 default;
        root /srv/ldphp/www/wildcard;

        location / {
            try_files $uri $uri/ /index.php?$args;
        }

        location ~ \.php$ {
            fastcgi_split_path_info ^(.+\.php)(/.+)$;
            include /etc/nginx/fastcgi_params;
            fastcgi_index index.php;
            fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_nam
            fastcgi_pass php;
        }
    }
}
```

Benchmark Config: supervisord.conf

```
[program:nginx]
command=nginx -c /srv/nginx.conf

[program:gold1]
command=/root/bin/server -insecure=:8011
directory=/srv/data

[program:gold2]
command=/root/bin/server -insecure=:8012
directory=/srv/data

[program:gold3]
command=/root/bin/server -insecure=:8013
```

```
directory=/srv/data

[program:gold4]
command=/root/bin/server -insecure=:8014
directory=/srv/data

[program:gold5]
command=/root/bin/server -insecure=:8015
directory=/srv/data

[program:gold6]
command=/root/bin/server -insecure=:8016
directory=/srv/data

[program:gold7]
command=/root/bin/server -insecure=:8017
directory=/srv/data

[program:gold8]
command=/root/bin/server -insecure=:8018
directory=/srv/data

[fcgi-program:gold1fcgi]
command=/root/bin/server
socket=unix:///tmp/gold1.sock
directory=/srv/data

[fcgi-program:gold2fcgi]
command=/root/bin/server
socket=unix:///tmp/gold2.sock
directory=/srv/data

[fcgi-program:gold3fcgi]
command=/root/bin/server
socket=unix:///tmp/gold3.sock
directory=/srv/data

[fcgi-program:gold4fcgi]
command=/root/bin/server
socket=unix:///tmp/gold4.sock
directory=/srv/data

[fcgi-program:gold5fcgi]
command=/root/bin/server
socket=unix:///tmp/gold5.sock
```

```
directory=/srv/data

[fcgi-program:gold6fcgi]
command=/root/bin/server
socket=unix:///tmp/gold6.sock
directory=/srv/data

[fcgi-program:gold7fcgi]
command=/root/bin/server
socket=unix:///tmp/gold7.sock
directory=/srv/data

[fcgi-program:gold8fcgi]
command=/root/bin/server
socket=unix:///tmp/gold8.sock
directory=/srv/data

[program:gold1tls]
command=/root/bin/server -bind=:8441
directory=/srv/data

[program:gold2tls]
command=/root/bin/server -bind=:8442
directory=/srv/data

[program:gold3tls]
command=/root/bin/server -bind=:8443
directory=/srv/data

[program:gold4tls]
command=/root/bin/server -bind=:8444
directory=/srv/data

[program:gold5tls]
command=/root/bin/server -bind=:8445
directory=/srv/data

[program:gold6tls]
command=/root/bin/server -bind=:8446
directory=/srv/data

[program:gold7tls]
command=/root/bin/server -bind=:8447
directory=/srv/data
```

```
[program:gold8tls]
command=/root/bin/server -bind=:8448
directory=/srv/data

[program:ldpy1]
command=gunicorn --umask 0000 -b 0.0.0.0:8021 ld
directory=/srv/data
environment=PYTHONPATH=/srv/ldpy

[program:ldpy2]
command=gunicorn --umask 0000 -b 0.0.0.0:8022 ld
directory=/srv/data
environment=PYTHONPATH=/srv/ldpy

[program:ldpy3]
command=gunicorn --umask 0000 -b 0.0.0.0:8023 ld
directory=/srv/data
environment=PYTHONPATH=/srv/ldpy

[program:ldpy4]
command=gunicorn --umask 0000 -b 0.0.0.0:8024 ld
directory=/srv/data
environment=PYTHONPATH=/srv/ldpy

[program:ldpy5]
command=gunicorn --umask 0000 -b 0.0.0.0:8025 ld
directory=/srv/data
environment=PYTHONPATH=/srv/ldpy

[program:ldpy6]
command=gunicorn --umask 0000 -b 0.0.0.0:8026 ld
directory=/srv/data
environment=PYTHONPATH=/srv/ldpy

[program:ldpy7]
command=gunicorn --umask 0000 -b 0.0.0.0:8027 ld
directory=/srv/data
environment=PYTHONPATH=/srv/ldpy

[program:ldpy8]
command=gunicorn --umask 0000 -b 0.0.0.0:8028 ld
directory=/srv/data
environment=PYTHONPATH=/srv/ldpy

[program:ldpy1uwsgi]
```

```
command=/srv/uwsgi --need-plugin python -s /tmp/ldpy1.sock --need-app --dis
directory=/srv/data

[program:ldpy2uwsgi]
command=/srv/uwsgi --need-plugin python -s /tmp/ldpy2.sock --need-app --dis
directory=/srv/data

[program:ldpy3uwsgi]
command=/srv/uwsgi --need-plugin python -s /tmp/ldpy3.sock --need-app --dis
directory=/srv/data

[program:ldpy4uwsgi]
command=/srv/uwsgi --need-plugin python -s /tmp/ldpy4.sock --need-app --dis
directory=/srv/data

[program:ldpy5uwsgi]
command=/srv/uwsgi --need-plugin python -s /tmp/ldpy5.sock --need-app --dis
directory=/srv/data

[program:ldpy6uwsgi]
command=/srv/uwsgi --need-plugin python -s /tmp/ldpy6.sock --need-app --dis
directory=/srv/data

[program:ldpy7uwsgi]
command=/srv/uwsgi --need-plugin python -s /tmp/ldpy7.sock --need-app --dis
directory=/srv/data

[program:ldpy8uwsgi]
command=/srv/uwsgi --need-plugin python -s /tmp/ldpy8.sock --need-app --dis
directory=/srv/data

[fcgi-program:php1]
command=php-cgi
socket=unix:///tmp/php1.sock

[fcgi-program:php2]
command=php-cgi
socket=unix:///tmp/php2.sock

[fcgi-program:php3]
command=php-cgi
socket=unix:///tmp/php3.sock

[fcgi-program:php4]
command=php-cgi
```

```
socket=unix:///tmp/php4.sock

[fcgi-program:php5]
command=php-cgi
socket=unix:///tmp/php5.sock

[fcgi-program:php6]
command=php-cgi
socket=unix:///tmp/php6.sock

[fcgi-program:php7]
command=php-cgi
socket=unix:///tmp/php7.sock

[fcgi-program:php8]
command=php-cgi
socket=unix:///tmp/php8.sock
```

Benchmark Config: /etc/sysconfig/ipvsadm

```
# gold HTTP
-A -t 18.16.5.32:8010 -s lc
-a -t 18.16.5.32:8010 -r 18.16.5.32:8011
-a -t 18.16.5.32:8010 -r 18.16.5.32:8012
-a -t 18.16.5.32:8010 -r 18.16.5.32:8013
-a -t 18.16.5.32:8010 -r 18.16.5.32:8014
-a -t 18.16.5.32:8010 -r 18.16.5.32:8015
-a -t 18.16.5.32:8010 -r 18.16.5.32:8016
-a -t 18.16.5.32:8010 -r 18.16.5.32:8017
-a -t 18.16.5.32:8010 -r 18.16.5.32:8018

# ldpy-gunicorn
-A -t 18.16.5.32:8020 -s lc
-a -t 18.16.5.32:8020 -r 18.16.5.32:8021
-a -t 18.16.5.32:8020 -r 18.16.5.32:8022
-a -t 18.16.5.32:8020 -r 18.16.5.32:8023
-a -t 18.16.5.32:8020 -r 18.16.5.32:8024
-a -t 18.16.5.32:8020 -r 18.16.5.32:8025
-a -t 18.16.5.32:8020 -r 18.16.5.32:8026
-a -t 18.16.5.32:8020 -r 18.16.5.32:8027
-a -t 18.16.5.32:8020 -r 18.16.5.32:8028

# gold TLS
-A -t 18.16.5.32:8440 -s lc
-a -t 18.16.5.32:8440 -r 18.16.5.32:8441
```

```
-a -t 18.16.5.32:8440 -r 18.16.5.32:8442
-a -t 18.16.5.32:8440 -r 18.16.5.32:8443
-a -t 18.16.5.32:8440 -r 18.16.5.32:8444
-a -t 18.16.5.32:8440 -r 18.16.5.32:8445
-a -t 18.16.5.32:8440 -r 18.16.5.32:8446
-a -t 18.16.5.32:8440 -r 18.16.5.32:8447
-a -t 18.16.5.32:8440 -r 18.16.5.32:8448
```

Benchmark NOOP Server in Go

```
package main

import (
        "flag"
        "net/http"
)

var (
        bind = flag.String("bind", "", "listening address")
)

func init() {
        flag.Parse()
}

func main() {
        http.ListenAndServe(*bind, nil)
}
```