Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής
και Υπολογιστών

Thesis subject

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΟΝΟΜΑ ΦΟΙΤΗΤΗ

Επιβλέπων : Υπεύθυνος Διπλωματικής
Τίτλος Υπευθύνου

Αθήνα, Σεπτέμβριος 9999

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής
και Υπολογιστών

Thesis subject

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΟΝΟΜΑ ΦΟΙΤΗΤΗ

Επιβλέπων : Υπεύθυνος Διπλωματικής
Τίτλος Υπευθύνου

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9η Σεπτεμβρίου 9999.

.....	.....	.....
Πρώτο μέλος επιτροπής Τίτλος μέλους	Δεύτερο μέλος επιτροπής Τίτλος μέλους	Τρίτο μέλος επιτροπής Τίτλος μέλους

Αθήνα, Σεπτέμβριος 9999

……

Όνομα Φοιτητή

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Περίληψη της διπλωματικής.

Λέξεις κλειδιά

Λέξη-κλειδί 1, λέξη-κλειδί 2, λέξη-κλειδί 3

Abstract

Abstract of diploma thesis.

Key words

Key-word 1, Key-word 2, Key-word 3

Ευχαριστίες

Ευχαριστίες.

Όνομα Φοιτητή,

Αθήνα, 9η Σεπτεμβρίου 9999

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-*-*, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Σεπτέμβριος 9999.

URL: http://www.softlab.ntua.gr/techrep/
FTP: ftp://ftp.softlab.ntua.gr/pub/techrep/

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction/Motivation

Bla-bla...

1.2 Thesis structure

Chapter 2: We define what ”cloud” means and mention some of the most notable examples. Then, we give a brief overview of the synnefo implementation, its key characteristics and why it can have a place in the current cloud world.

Chapter 3: We present the architecture of Archipelago and provide the necessary theoretical background (mmap, IPC) the reader needs to understand its basic concepts. Then, we thoroughly explain how Archipelago handles I/O requests. Finally, we mention what are the current storage mechanisms for Archipelago and evaluate their performance.

Chapter 4: We explain why tiering is important and what is the state of tiered storage at the moment (bcache, flashcache, memcached, ramcloud, couchbase). Then, we provide the related theoretical background for cached (hash-tables, LRUs). Finally, we defend why we chose to roll out our own implementation.

Chapter 5: We explain the design of cached, the building blocks that is consisted of (xcache, xworkq, xwaitq). Then, we provide extensive benchmark results and compare them to the ones of Chapter 3.

Chapter ??: TODO

Chapter ??: We draw some concluding remarks and propose some future work.

Chapter 2

Chapter 2

2.1 Section 1

This section has an important citation[aeal99]

2.1.1 Subsection 1

This subsection has code in Haskell:

```
foo [] = []
foo h:t = 9: foo t
```

Listing 2.1: Sample code

It also has a list:

Item 1 First item

Item 2 Second item and a footnote[1].

Item 3 Third item and text in *italics*.

And an enumerated list:

1. First item.
2. Second item and text in **bold**

2.2 Section 2

2.2.1 Subsection 1

This subsection has a link to the block of code 2.1 in Section 1.

2.2.2 Subsection 2

This subsection has a FIXME comment, visible only to the author.

[1] Footnote description.

Chapter 3

3.1 Section 1

This how we add a url: http://www.example.org

3.2 Section 2

And this is how we point to Figure 3.1.

Figure 3.1: This is an image

3.2.1 Sub-section 3

Another way to create a list:

- Item 1

- Item 2
- Item 3
- Item 4

Chapter 4

4.1 Section 1

We can also use a special fonts to differentiate between text and $math()$.

Chapter 5

Design of cached

5.1 General

Currently, Archipelago doesn’t have an xseg peer that can act as a cache for block requests. Having a cache however is essential for a tiered storage and cached peer is implemented for this reason.

5.2 Implementation details

cached is based on xseg and also on the following xtypes:

- **xcache** (for cache support) * xwork (for job support) * xworkq (for atomicity in execution of jobs) * xwaitq (for conditional execution of jobs)

More specifically, cached consists of the cache provided by xcache and a pre-allocated number of objects. An object is divided in buckets and its size, as well as bucket size, are defined by the user.

The fact that objects are pre-allocated means two things:

1) We don’t need to care about memory fragmentation and system call overhead 2) We cannot index single buckets. <FILLME>

5.3 The xcache xtype

5.4 The xworkq xtype

Every object has a workq. Whenever a new request is accepted/received for an object, it is enqueued in the workq and we are sure that only one thread at a time can have access to the objects data and metadata.

For more information, see the xworkq.

5.5 The xwaitq xtype

When a thread tries to insert an object in cache but fails, due to the fact that cache is full, the request is enqueued in the xcache waitq, which is signaled every time an object is freed.

For more information, see the xwaitq.

5.6 Cached internals

5.6.1 Object states

Every object has a state, which is set atomically by threads. The state list is the following:

* READY: the object is ready to be used * FLUSHING: the object is flushing its dirty buckets * DELETING: there is a delete request that has been sent to the blocker for this object * INVALIDATED: the object has been deleted * FAILED: something went very wrong with this object

Also, object buckets have their own states too:

* INVALID: the same as empty * LOADING: there is a pending read to blocker for this bucket * VALID: the bucket is clean and can be read * DIRTY: the bucket can be read but its contents have not been written to the underlying storage * WRITING: there is a pending write to blocker for this bucket

Finally, for every object there are bucket state counters, which are increased/ decreased when a bucket state is changed. These counters give us an O(1) glimpse to the bucket states of an object.

5.6.2 Per-object peer requests

Reads and writes to objects are practically read/write request from other peers, for which a peer request has been allocated. There are cases though when an object has to allocate its own peer request e.g. due to a flushing of its dirty buckets. Since this must be fast, there are pre-allocated requests hard-coded in the struct of each object which can be used in such cases.

5.6.3 Write policy

The user must define beforehand what is the write policy of cache. There are two options: writethrough and writeback. On a side note, as far as reads and cache misses are concerned, cached operates under a write-allocate policy.

Bibliography

[aeal99] Some author et al., “Name of citation”, in *Proceedings of the 99th ACM Symposium on Something (POPL’99)*, pp. 999–999, Nine, 9999.