Иерархические файловые системы мертвы.

Автор:Сельцер, Марго и Николас Мерфи. Гарвард.
Дата публикации: 21-ое сентября 2011 год
Ссылка: https://dash.harvard.edu/handle/1/5136361

Краткое содержание статьи

1 Введение

Можно выделить две структуры хранения данных: файловую систему и базу данных. Использование базы данных оправдано в том случае если есть необходимость задания сложных зависимостей между разными видами данных, а также когда прямой доступ к данным необязателен. В ином случае использование базы данных для хранения информации не рационально, вместо нее используется более простая файловая система. Но с момента появления первой файловой системы прошло достаточно много времени, и сейчас вместо 300 Мб жесткого диска появились 300 Гб накопители. А файлы не так далеко ушли от размера своих предшественников. Все это позволило пользователю вмещать гигабайты медиа на одном персональном компьютере.
Благодаря поисковым системам (такими как Google или MacOS Spotlight) мы знакомы с понятием, что такое тег. Но поисковые системы основаны на базе данных, и поэтому обладают соответствующими недостатками, такими как слишком громоздкая реализация, не оптимизированы под определенные задачи или по-просту сложны в управлении и эксплуатации.
Многие авторы публицистическо-научной литературы уже давно пытаются привлечь интерес читателей к комбинированию файловой системы с базами данных.[2][1]

Иерархические файловые системы имеют следующие недостатки:

- Несоответствие файлов и директорий(пользователь испытывает затруднения при поиске файлов)
- Ограниченность систематизации файлов(чтобы организация информации была полной, должна быть возможность поместить один объект в несколько категорий, а этого текущая файловая система не позволяет)
- Низкая производительность (существуют так называемые хот-споты [1] и медленный поиск)

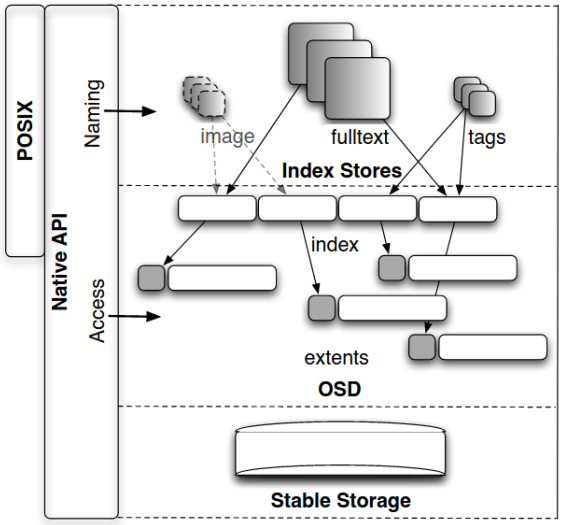
В основу современной файловой системы должно входить следующее:

- Обратная совместимость
- Наименование отделено от адреса (наименование не должно описывать где храниться файл)
- Независимость данных (каждое приложение должно обрабатывать файлы по разному, потому что файловая система рассматривает файлы как необработанную байтовую строку)
- Прямой доступ к информации

[1] Хот-спот - это участок кода в программе, на который приходится бóльшая часть исполняемых инструкций процессора

2 Файловая система hFAD

hFad(Hierarchical Filesystems are Dead) - файловая система основанная на объектах, но в отличие от других OSD[2], каждый байт объекта доступен для изменения пользователем и прямой доступ к объектам реализован в виде ID объекта.



hFAD API состоит из двух компонентов: интерфейса наименования и интерфейса доступа. Интерфейс наименования идентифицирует объект по теговым именам(определяет индекс, а также то, как нужно обработать файл), а затем передает управление интерфейсу доступа.

2.1 Интерфейс наименования

Имя объекта идентифицируется по одной или более паре тег/значение. По тегу можно определить как нужно обрабатывать значение и то, в каком индексе расположен объект. Операция поиска имени может возвращать несколько объектов. Более того, не нужно, чтобы объект в запросе был уникальным. Необходимо лишь чтобы идентификатор

[2] OSD (Object-based storage device)

информации на OSD уровне был уникальным(ID объекта).

Использование	Тег	Значение
POSIX	POSIX	pathname
Поиск	FULLTEXT	term
В ручную	USER	logname
	UDEF	annotations
Приложение	APP	applicationname
FastPath	ID	object identifier
	USER	logname

2.2 Интерфейс доступа

В то время пока наша файловая система предоставляет полный доступ к байтам, хранящимся на диске, нам нужен способ вставлять байты по середине или обрезать информацию из любого места в файле. Согласно стандарту POSIX мы добавим insert и truncate функции.

2.2.1 Индексное хранилище

Эффективность подобных систем зависит в основном от правильного способа получения ID объекта. Например, база данных, хранящая информацию в виде ключ/значение не подойдет для хранения fulltext значений и в тоже время fulltext индексное пространство не подойдет для хранения ключ/значение, и по той же самой причине оно не подойдет для хранения изображений. Так что различные индексные пространства это первая необходимость в современной файловой системе.

2.2.2 OSD слой

Слой представляет собой абстракцию уникальных контейнеров байт. Каждый контейнер(объект) имеет свои атрибуты. Эта реализация OSD сравнима с ZFS Data Management Unit(DMU), но в отличии от DMU, OSD не предоставляет коллекции объектов(так называемые objset).

2.2.3 Реализация

Linux/FUSE берем за основу. Berkeley DB будет отвечать за предоставление абстракции над устройствами и за выделение памяти под хранилище. Мы используем Berkeley Data Base чтобы хранить все строковые индексы с мета-данными для объектов, в которой в качестве ключа будет использоваться по-байтовое смещение и в которой в качестве значение используются адреса в дисковом пространстве (полагаю что здесь автор имел ввиду CHS) и количество байт занимаемых файлом.

Ссылки

[1] “In Proceedings of the 13th ACM Symposium on Operating Systems Principles”. In: 1991, pp. 16–25.

[2] Olson. “In Proceedings of the 1993 USENIX Winter Conference”. In: 1993, pp. 205–218.