Ceph – petabyte scale storage

Karol Sobczak

Uniwersytet Warszawski

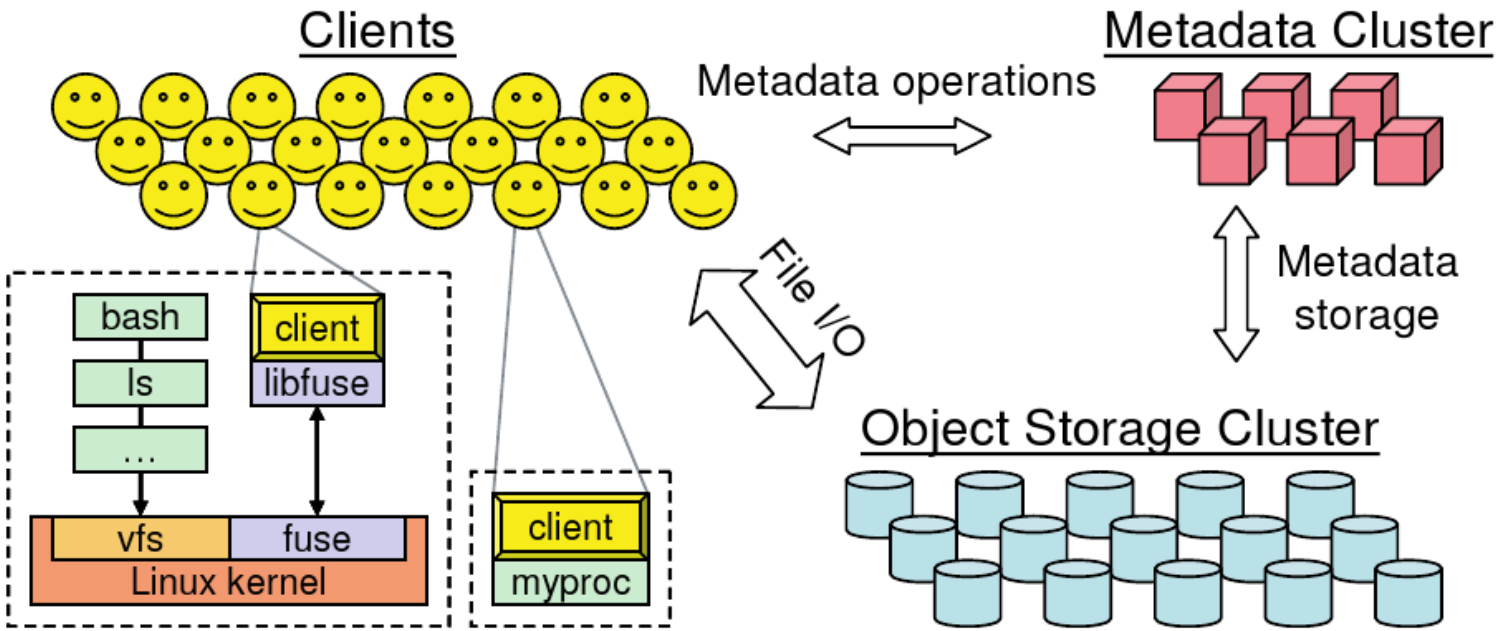
27 listopada 2008

1. Ceph należy do klasy rozproszonych systemów plików takich jak GoogleFS czy MooseFS.
2. Jest to system w pełni Open Source.
3. W przeciwieństwie do pozostałych systemów w Ceph metadane także są umieszczone na klastrze serwerów.
4. Brak list alokacji dla plików.
5. Zapewnia semantykę POSIX, ale także w pewnych kwestiach ją rozszerza.
6. W założeniach ma sobie dobrze radzić z sytuacjami, w których wiele (nawet tysiące) klientów próbuje uzyskać dostęp do tego samego pliku lub katalogu (tzn. hot spot).

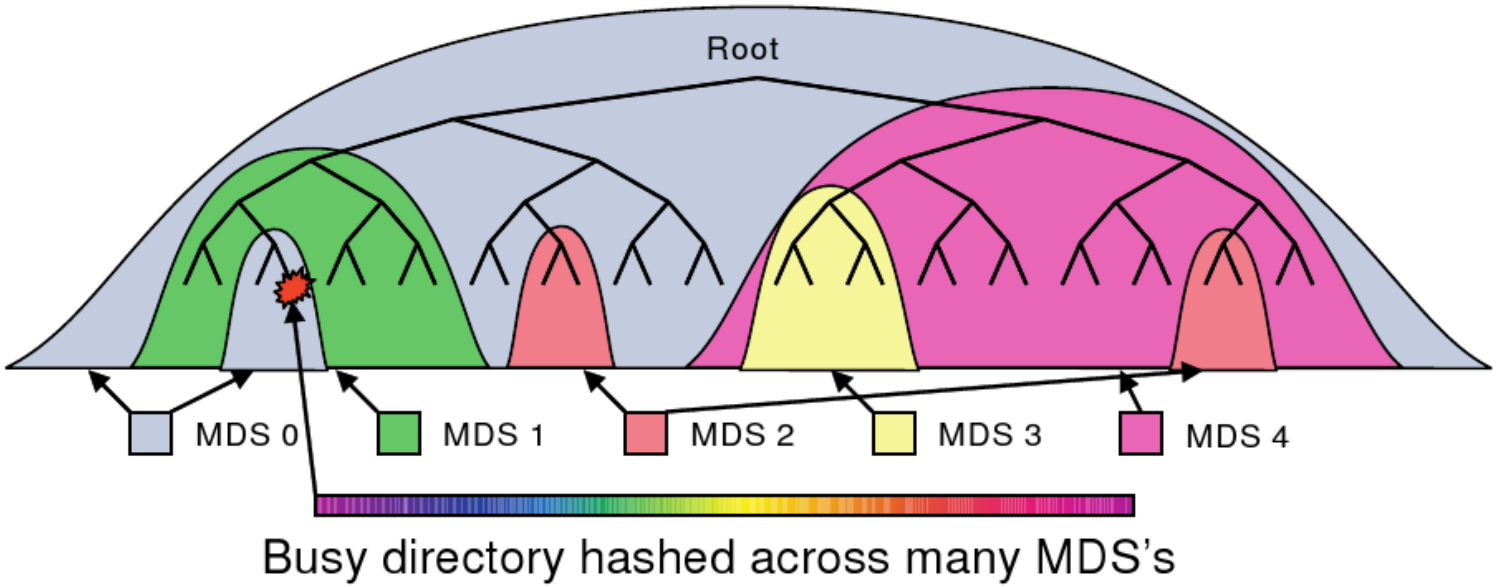
- Badania nad Ceph są prowadzone w SSRC (Storage System Research Center) będący organizacją należącą do University of California. Do sponsorów SSRS należą takie firmy jak Hitachi, Seagate, HP.
- Do twórców należą między innymi:
  - Sage A.Weil główny programista
  - Profesor Scott A. Brandt, Chair
  - Doktor Richard Golding
  - Profesor Charlie McDowell
  - Profesor Carlos Maltzahn
  - Profesor Ethan L. Miller
- System jest ciągle prototypem.

Budowę Ceph-u można podzielić na cztery główne części:

1. Object storage devices (OSD) cluster – komputery (zazwyczaj przeciętne i zawodne) przechowujące fragmenty plików. W Ceph każdy fragment pliku jest nazywany obiektem. OSD cluster jest to twór, który można znaleźć w innych tego typu systemach plików.
2. Metadata servers (MDS) cluster – komputery zarządzające metadanymi. Podział metadanych jest wybierany za pomocą nowatorskiej metody Dynamic Subtree Partitioning.
3. Monitors cluster – komputery nadzorujące pracę pozostałych komputerów.
4. Klient – biblioteka lub moduł FUSE udostępniający funkcjonalność Ceph.



Rysunek: Budowa Ceph (źródło: [2])



Rysunek: Hierarchia i podział metadanych między MDS-y (źródło: [2])

- Poszczególne kroki podczas otwierania iwęzła:

- Poszczególne kroki podczas otwierania iwęzła:
  1. Klient łączy się z najgłębszym znanym serwerem MDS na ścieżce.

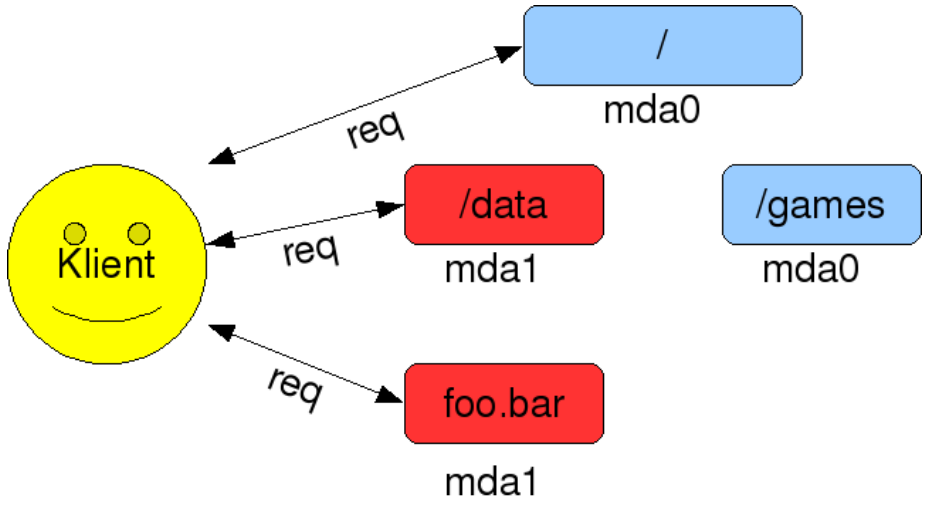
- Poszczególne kroki podczas otwierania iwęzła:
  1. Klient łączy się z najgłębszym znanym serwerem MDS na ścieżce.
  2. Klient odpytuje kolejne serwery na ścieżce do pliku.

- Poszczególne kroki podczas otwierania iwęzła:
  1. Klient łączy się z najgłębszym znanym serwerem MDS na ścieżce.
  2. Klient odpytuje kolejne serwery na ścieżce do pliku.
  3. Jeśli klient ma prawo do otwarcia danego pliku to zarządzający plikiem MDS zwraca klientowi numer iwęzła, rozmiar pliku i informację o strategii rozmieszczenia pliku potrzebną do odnalezienia poszczególnych fragmentów pliku.

- Poszczególne kroki podczas otwierania iwęzła:
  1. Klient łączy się z najgłębszym znanym serwerem MDS na ścieżce.
  2. Klient odpytuje kolejne serwery na ścieżce do pliku.
  3. Jeśli klient ma prawo do otwarcia danego pliku to zarządzający plikiem MDS zwraca klientowi numer iwęzła, rozmiar pliku i informację o strategii rozmieszczenia pliku potrzebną do odnalezienia poszczególnych fragmentów pliku.
  4. Ponadto serwer MDS może dać klientowi prawo do wykonywania określonych operacji (obsługiwane operacje to: odczyt, buforowany odczyt, zapis, buforowany zapis).

- Poszczególne kroki podczas otwierania iwęzła:
  1. Klient łączy się z najgłębszym znanym serwerem MDS na ścieżce.
  2. Klient odpytuje kolejne serwery na ścieżce do pliku.
  3. Jeśli klient ma prawo do otwarcia danego pliku to zarządzający plikiem MDS zwraca klientowi numer iwęzła, rozmiar pliku i informację o strategii rozmieszczenia pliku potrzebną do odnalezienia poszczególnych fragmentów pliku.
  4. Ponadto serwer MDS może dać klientowi prawo do wykonywania określonych operacji (obsługiwane operacje to: odczyt, buforowany odczyt, zapis, buforowany zapis).
- Umiejscowienie poszczególnych kawałków pliku w klastrze otrzymujemy domyślnie za pomocą pseudolosowej funkcji CRUSH (brak list alokacji na serwerze metadanych).

- Poszczególne kroki podczas otwierania iwęzła:
  1. Klient łączy się z najgłębszym znanym serwerem MDS na ścieżce.
  2. Klient odpytuje kolejne serwery na ścieżce do pliku.
  3. Jeśli klient ma prawo do otwarcia danego pliku to zarządzający plikiem MDS zwraca klientowi numer iwęzła, rozmiar pliku i informację o strategii rozmieszczenia pliku potrzebną do odnalezienia poszczególnych fragmentów pliku.
  4. Ponadto serwer MDS może dać klientowi prawo do wykonywania określonych operacji (obsługiwane operacje to: odczyt, buforowany odczyt, zapis, buforowany zapis).
- Umiejscowienie poszczególnych kawałków pliku w klastrze otrzymujemy domyślnie za pomocą pseudolosowej funkcji CRUSH (brak list alokacji na serwerze metadanych).
- Aktualnie serwery wierzą wszystkim klientom.



Rysunek: Schemat komunikacji przy otwieraniu pliku

Jak wygląda podział pliku w systemie?

Jak wygląda podział pliku w systemie?

- Każdy plik jest podzielony na części (tak zwane Objects).

Jak wygląda podział pliku w systemie?

- Każdy plik jest podzielony na części (tak zwane Objects).
- Nazywanie poszczególnych obiektów pliku jest bardzo proste: nazwa obiektu łączy w sobie numer iwęzła oraz numer części pliku (dlatego klient otrzymuje informację o wielkości pliku przy jego otwieraniu).

Jak wygląda podział pliku w systemie?

- Każdy plik jest podzielony na części (tak zwane Objects).
- Nazywanie poszczególnych obiektów pliku jest bardzo proste: nazwa obiektu łączy w sobie numer iwęzła oraz numer części pliku (dlatego klient otrzymuje informację o wielkości pliku przy jego otwieraniu).
- OSD są znajdowane przy pomocy funkcji CRUSH na podstawie nazwy obiektu.

Jak wygląda podział pliku w systemie?

- Każdy plik jest podzielony na części (tak zwane Objects).
- Nazywanie poszczególnych obiektów pliku jest bardzo proste: nazwa obiektu łączy w sobie numer iwęzła oraz numer części pliku (dlatego klient otrzymuje informację o wielkości pliku przy jego otwieraniu).
- OSD są znajdowane przy pomocy funkcji CRUSH na podstawie nazwy obiektu.
- Obiekty przydzielane są w razie potrzeby. Brak danego obiektu oznacza, że w danym fragmencie pliku są same zera.

Co się dzieje w przypadu gdy wielu klientów chce czytać i pisać dany plik?

Co się dzieje w przypadu gdy wielu klientów chce czytać i pisać dany plik?

- Semantyka POSIX mówi nam, że zapisy są atomowe oraz kolejne odczyty powinny uwzględniać dokonane już zapisy.

Co się dzieje w przypadu gdy wielu klientów chce czytać i pisać dany plik?

- Semantyka POSIX mówi nam, że zapisy są atomowe oraz kolejne odczyty powinny uwzględniać dokonane już zapisy.
- Aby zachować tą semantykę, serwer zarządzający metadanymi pliku odbiera prawo do buforowania zapisów oraz odczytów od wszystkich klientów, którzy otworzyli dany plik.

Co się dzieje w przypadu gdy wielu klientów chce czytać i pisać dany plik?

- Semantyka POSIX mówi nam, że zapisy są atomowe oraz kolejne odczyty powinny uwzględniać dokonane już zapisy.
- Aby zachować tą semantykę, serwer zarządzający metadanymi pliku odbiera prawo do buforowania zapisów oraz odczytów od wszystkich klientów, którzy otworzyli dany plik.
- Dodatkowo klienci przy zapisywaniu danych do wielu obiektów będą w takiej sytuacji zakładać blokady na OSD, aby po zapisaniu danych natychmiast je zwolnić.

Czy naprawdę semantyka POSIX jest nam potrzebna w każdym przypadku?

Czy naprawdę semantyka POSIX jest nam potrzebna w każdym przypadku?

- Możliwość wyłączenia takiej synchronizacji poprzez globalny switch.

Czy naprawdę semantyka POSIX jest nam potrzebna w każdym przypadku?

- Możliwość wyłączenia takiej synchronizacji poprzez globalny switch.
- Propozycja rozszerzenia standardu POSIX:
  - O_LAZY – rozluźnia semantykę POSIX przy otwieraniu pliku (klient nie rezygnuje z buforowania danych oraz atrybutów pliku).
  - lazyio_propagate – wysłanie określonych danych z bufora do rzeczywistego pliku.
  - lazyio_synchronize – zapewnia, że zmiany dokonane w rzeczywistym pliku zostaną odzwierciedlone w przyszłych odczytach.

Kolejnymi kłopotliwymi operacjami są operacje odczytywania zawartości katalogów (funkcja readdir) oraz statystyk plików (funkcja stat).

Kolejnymi kłopotliwymi operacjami są operacje odczytywania zawartości katalogów (funkcja readdir) oraz statystyk plików (funkcja stat).

- Jako, że operacje stat są często następstwem operacji readdir, to klient przy wykonywaniu operacji readdir otrzymuje od razu wyniki funkcji stat dla plików katalogu. Jeśli teraz aplikacja wykona kilka wywołań stat to klient zwraca już otrzymane dane, inaczej dodatkowe dane są odrzucane.

Kolejnymi kłopotliwymi operacjami są operacje odczytywania zawartości katalogów (funkcja readdir) oraz statystyk plików (funkcja stat).

- Jako, że operacje stat są często następstwem operacji readdir, to klient przy wykonywaniu operacji readdir otrzymuje od razu wyniki funkcji stat dla plików katalogu. Jeśli teraz aplikacja wykona kilka wywołań stat to klient zwraca już otrzymane dane, inaczej dodatkowe dane są odrzucane.
- Nowa propozycja rozszerzenia POSIX: funkcja readdirplus – wykonuje jawnie wczytanie zawartości katalogu i statystyk plików.

Co w przypadku, gdy jakiś klient wywołuje stat na pliku otwartym do pisania przez innych klientów?

Co w przypadku, gdy jakiś klient wywołuje stat na pliku otwartym do pisania przez innych klientów?

- Ceph odbiera na chwilę prawa do zapisu do pliku od wszystkich klientów i zbiera od nich dane odnośnie wielkości pliku oraz mtime. Jest to niezbędne dla zachowania semantyki POSIX.

Co w przypadku, gdy jakiś klient wywołuje stat na pliku otwartym do pisania przez innych klientów?

- Ceph odbiera na chwilę prawa do zapisu do pliku od wszystkich klientów i zbiera od nich dane odnośnie wielkości pliku oraz mtime. Jest to niezbędne dla zachowania semantyki POSIX.
- Kolejna propozycja funkcji: statlite – pozwala na określenie, które statystyki pliku nas interesują (być może nie ma konieczności blokowania zapisów).

- Zawartość katalogu przechowywana jest w pojedynczym obiekcie o nazwie odpowiadającej numerowi iwęzła katalogu.

- Zawartość katalogu przechowywana jest w pojedynczym obiekcie o nazwie odpowiadającej numerowi iwęzła katalogu.
- Iwęzły plików są przechowywane w strukturze obiektu katalogu, dzięki czemu można szybko uzyskać dane o plikach w katalogu.

- Zawartość katalogu przechowywana jest w pojedynczym obiekcie o nazwie odpowiadającej numerowi iwęzła katalogu.
- Iwęzły plików są przechowywane w strukturze obiektu katalogu, dzięki czemu można szybko uzyskać dane o plikach w katalogu.
- Pojedyńczy MDS posiada dziennik do którego zapisuje zmiany w metadanych.

Pozostaje problem twardych dowiązań (katalogi zawierają węzły zamiast wskaźników do węzłów). Jeden katalog zawiera właściwego węzła (primary dentry), a pozostałe katalogi zawierające ten sam plik trzymają tylko numer węzła (remote dentry). Wiedzą, gdzie jest właściwy węzeł za pomocą specjalnej struktury anchor table (utrzymywanej przez pojedyńczego MDS-a dla zduplikowanych instancji pliku):

Path	Inode	Parent	Ref
/usr/bin/nano	123	12	1
/usr/bin	12	10	1
/usr/lib/ld.so	456	11	1
/usr/lib	11	10	1
/usr	10	1	2
/	1		1

Rysunek: Anchor table (źródło: [3])

- MDS może być repliką lub właścicielem konkretnej porcji przechowywanych w pamięci metadanych.

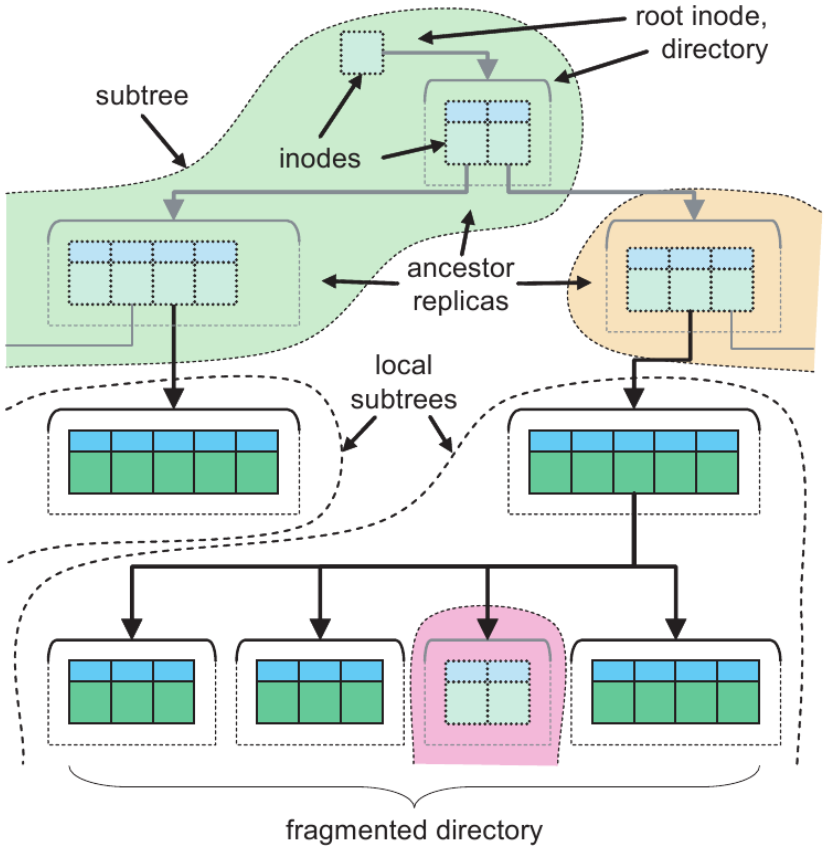
- MDS może być repliką lub właścicielem konkretnej porcji przechowywanych w pamięci metadanych.
- Wszystkie przechowywane przez MDS metadane muszą być połączone (niekoniecznie bezpośrednio) z korzeniem hierarchii metadanych. Oznacza to, że każdy MDS replikuje metadane przodków aż do korzenia.

- MDS może być repliką lub właścicielem konkretnej porcji przechowywanych w pamięci metadanych.
- Wszystkie przechowywane przez MDS metadane muszą być połączone (niekoniecznie bezpośrednio) z korzeniem hierarchii metadanych. Oznacza to, że każdy MDS replikuje metadane przodków aż do korzenia.
- Wszystkie metadane mają swojego właściciela.

- MDS może być repliką lub właścicielem konkretnej porcji przechowywanych w pamięci metadanych.
- Wszystkie przechowywane przez MDS metadane muszą być połączone (niekoniecznie bezpośrednio) z korzeniem hierarchii metadanych. Oznacza to, że każdy MDS replikuje metadane przodków aż do korzenia.
- Wszystkie metadane mają swojego właściciela.
- Jeśli jakiś MDS przechowuje porcję metadanych, to zawsze wie, kto jest jej właścicielem.

- MDS może być repliką lub właścicielem konkretnej porcji przechowywanych w pamięci metadanych.
- Wszystkie przechowywane przez MDS metadane muszą być połączone (niekoniecznie bezpośrednio) z korzeniem hierarchii metadanych. Oznacza to, że każdy MDS replikuje metadane przodków aż do korzenia.
- Wszystkie metadane mają swojego właściciela.
- Jeśli jakiś MDS przechowuje porcję metadanych, to zawsze wie, kto jest jej właścicielem.
- Właściciel metadanych zna wszystkie repliki.

- MDS może być repliką lub właścicielem konkretnej porcji przechowywanych w pamięci metadanych.
- Wszystkie przechowywane przez MDS metadane muszą być połączone (niekoniecznie bezpośrednio) z korzeniem hierarchii metadanych. Oznacza to, że każdy MDS replikuje metadane przodków aż do korzenia.
- Wszystkie metadane mają swojego właściciela.
- Jeśli jakiś MDS przechowuje porcję metadanych, to zawsze wie, kto jest jej właścicielem.
- Właściciel metadanych zna wszystkie repliki.
- Do zachowania spójności używany jest skomplikowany mechanizm blokad. Operacje modyfikujące metadane są kierowane do właściciela.



Rysunek: Metadane pojedyńczego MDS-a (źródło: [3])

- Przez poszczególne MDS-y utrzymywany jest miernik popularności metadanych (liczniki odczytów, zapisów, etc.). Co jakiś czas MDS-y wymieniają się statystykami.

- Przez poszczególne MDS-y utrzymywany jest miernik popularności metadanych (liczniki odczytów, zapisów, etc.). Co jakiś czas MDS-y wymieniają się statystykami.
- Wymiana odpowiedzialności za poddrzewo metadanych pomiędzy dwoma MDS-ami jest zrealizowana na bazie transakcji.

Obiekty składające się na całość pliku są przydzielane do Placement Groups (PGs). Każde PG reprezentuje zestaw obiektów, które są replikowane przez zbiór maszyn OSD.

- **pgid=(r,hash(o)&m)**, gdzie $m = 2^k - 1$ oraz **m** jest ustalone globalnie dla całego systemu.

Obiekty składające się na całość pliku są przydzielane do Placement Groups (PGs). Każde PG reprezentuje zestaw obiektów, które są replikowane przez zbiór maszyn OSD.

- **pgid=(r,hash(o)&m)**, gdzie $m = 2^k - 1$ oraz **m** jest ustalone globalnie dla całego systemu.
- Aby zapewnić skalowalność systemu używa się jednak **pgid=(r,stablemod(hash(o),n,m))**.

```
procedure STABLEMOD(x, n, m)
    if x&m < n then
        return x&m
    else
        return x&(m >> 1)
    end if
end
```

Korzyści? Przy dodaniu bitu do wartości **m** nie musimy od razu dzielić wszystkich PG na pół i przenosić danych. Możemy robić to stopniowo.

CRUSH (Controlled Replication Under Scalable Hashing) – funkcja haszująca PGs w listę maszyn, na których powinny znajdować się repliki obiektu, którym zainteresowany jest klient. Do obliczenia wyniku funkcja CRUSH potrzebuje:

- Mapy klastra (cluster map) – hierarchiczny opis klastra maszyn OSD uwzględniający takie czynniki jak fizyczne położenie maszyn.
- Reguł rozmieszczania (placement rules) – reguły rozmieszczania replik dla obiektów.
- Nazwy PG.

Cluster map

- Mapa składa się z urządzeń (OSD devices) oraz kubełków (buckets). Każde urządzenie oraz kubełek ma identyfikator oraz związaną z nim wagę.
- Każdy kubełek może zawierać dowolną liczbę innych kubełków lub urządzeń. Waga kubełka jest sumą wag elementów które zawiera.
- Wagi urządzeń ustala administrator w celu określenia za jak dużą porcję danych odpowiada dane urządzenie.

Taka budowa mapy pozwala na wymodelowanie całkiem skomplikowanych zależności pomiędzy maszynami w klastrze. Można na przykład zdefiniować “półkę” takich samych maszyn OSD, następnie “szafę” jako zbiór “półek”, następnie “szafy” można pogrupować w “pokoje”. Dzięki napisaniu odpowiednich reguł rozmieszczania replik można zminimalizować skutki awarii (na przykład braku zasilania).

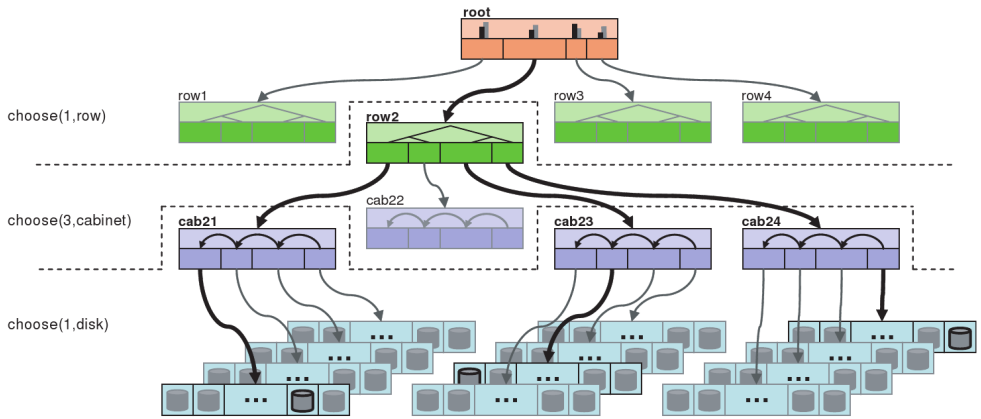
Placement rules

Proste operacje są aplikowane do każdego poziomu hierarchii w mapie klastra:

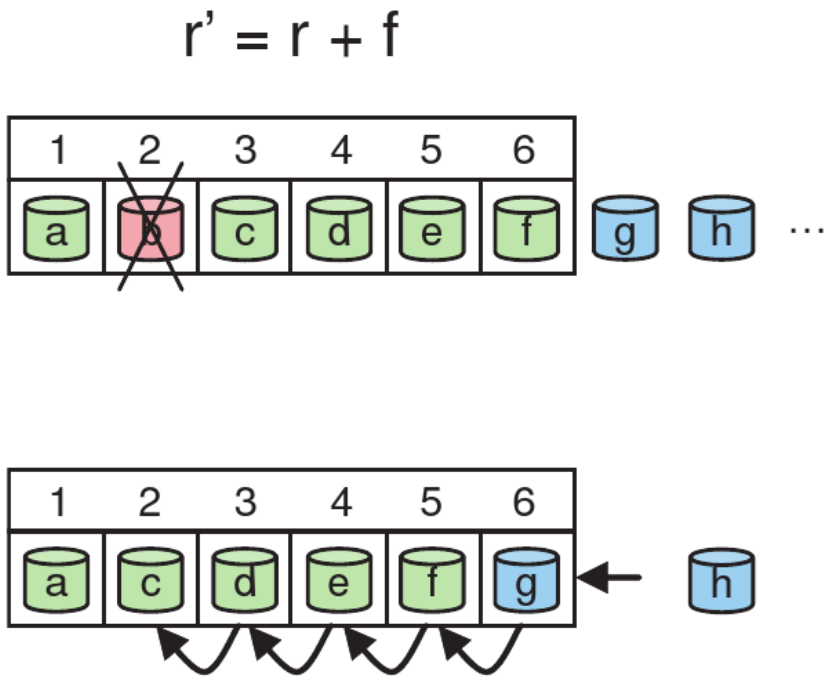
- **select(n,t)** – stworzenie nowej listy roboczej poprzez zmapowanie elementów ze starej listy roboczej w taki sposób, że dla każdego elementu starej listy roboczej wybieramy **n** dzieci typu **t**. Wybór nie może się powtarzać.
- **take(a)** – wybranie obiektu z mapy i wrzucenie go do listy roboczej. Używana przy starcie algorytmu.
- **emit** – zwrócenie wyniku. Używana przy końcu algorytmu.

Action	Resulting $\vec{i}$
take(root)	root
select(1,row)	row2
select(3,cabinet)	cab21 cab23 cab24
select(1,disk)	disk2107 disk2313 disk2437
emit	

Rysunek: Przykładowa reguła rozmieszczania (źródło: [3])



Rysunek: Przykładowa mapa (źródło: [3])



Rysunek: Rozwiązywanie konfliktów (źródło: [3])

Skąd poszczególne maszyny znają mapę oraz reguły rozmieszczania?

Skąd poszczególne maszyny znają mapę oraz reguły rozmieszczania?

- Mapa oraz reguły rozmieszczania składają się na OSD Cluster Map.

Skąd poszczególne maszyny znają mapę oraz reguły rozmieszczania?

- Mapa oraz reguły rozmieszczania składają się na OSD Cluster Map.
- Klaster monitorów obserwuje sieć i wprowadza zmiany do mapy (zbiorowo zarządzają mapą).

Skąd poszczególne maszyny znają mapę oraz reguły rozmieszczania?

- Mapa oraz reguły rozmieszczania składają się na OSD Cluster Map.
- Klaster monitorów obserwuje sieć i wprowadza zmiany do mapy (zbiorowo zarządzają mapą).
- Administrator może wprowadzić zmiany do mapy i reguł rozmieszczania.

Skąd poszczególne maszyny znają mapę oraz reguły rozmieszczania?

- Mapa oraz reguły rozmieszczania składają się na OSD Cluster Map.
- Klaster monitorów obserwuje sieć i wprowadza zmiany do mapy (zbiorowo zarządzają mapą).
- Administrator może wprowadzić zmiany do mapy i reguł rozmieszczania.
- Przy komunikacji maszyny wymieniają się numerami wersji OSD Cluster Map i jeśli któraś maszyna ma starszą wersję, to przekazywane są jej zmiany w OSD Cluster Map.

Replikacja danych opiera się na technice replikacji głównej kopii danych.

Replikacja danych opiera się na technice replikacji głównej kopii danych.

- Dane są replikowanie w obrębie jednej Placement Group (PG).

Replikacja danych opiera się na technice replikacji głównej kopii danych.

- Dane są replikowanie w obrębie jednej Placement Group (PG).
- Wszystkie maszyny OSD w obrębie jednego PG są posortowane.

Replikacja danych opiera się na technice replikacji głównej kopii danych.

- Dane są replikowanie w obrębie jednej Placement Group (PG).
- Wszystkie maszyny OSD w obrębie jednego PG są posortowane.
- Klienci kierują wszystkie zapisy do pierwszego sprawnego OSD z listy (Primary OSD).

Replikacja danych opiera się na technice replikacji głównej kopii danych.

- Dane są replikowanie w obrębie jednej Placement Group (PG).
- Wszystkie maszyny OSD w obrębie jednego PG są posortowane.
- Klienci kierują wszystkie zapisy do pierwszego sprawnego OSD z listy (Primary OSD).
- Wszystkie odczyty są kierowane do głównego OSD (Primary OSD).

Zapis nowej porcji danych:

Zapis nowej porcji danych:

- Klient wysyła porcję zmodyfikowanego obiektu do głównego OSD.

Zapis nowej porcji danych:

- Klient wysyła porcję zmodyfikowanego obiektu do głównego OSD.
- Główny OSD przesyła dane do pozostałych replik.

Zapis nowej porcji danych:

- Klient wysyła porcję zmodyfikowanego obiektu do głównego OSD.
- Główny OSD przesyła dane do pozostałych replik.
- Repliki po umieszczeniu danych w buforze wysyłają do głównego OSD komunikat **ack** (nie oznacza to jeszcze zapisania danych na dysku).

Zapis nowej porcji danych:

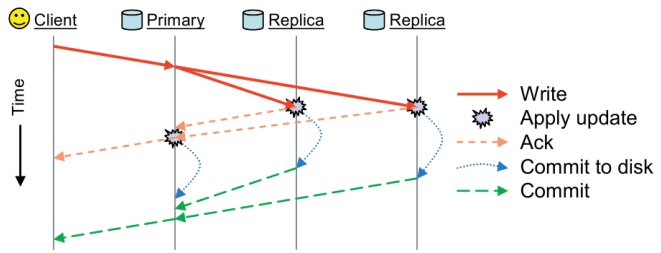
- Klient wysyła porcję zmodyfikowanego obiektu do głównego OSD.
- Główny OSD przesyła dane do pozostałych replik.
- Repliki po umieszczeniu danych w buforze wysyłają do głównego OSD komunikat **ack** (nie oznacza to jeszcze zapisania danych na dysku).
- Po otrzymaniu wszystkich ack główny OSD sam buforuje dane do zapisu i wysyła **ack** do klienta.

Zapis nowej porcji danych:

- Klient wysyła porcję zmodyfikowanego obiektu do głównego OSD.
- Główny OSD przesyła dane do pozostałych replik.
- Repliki po umieszczeniu danych w buforze wysyłają do głównego OSD komunikat **ack** (nie oznacza to jeszcze zapisania danych na dysku).
- Po otrzymaniu wszystkich ack główny OSD sam buforuje dane do zapisu i wysyła **ack** do klienta.
- We wszystkich replikach zmodyfikowany obiekt posiada nowy, wyższy numer wersji. Dodawany jest także wpis do dziennika PG na każdym OSD (który także posiada numer wersji). Wersje są ustalane przez główny OSD.

Zapis nowej porcji danych – ciąg dalszy:

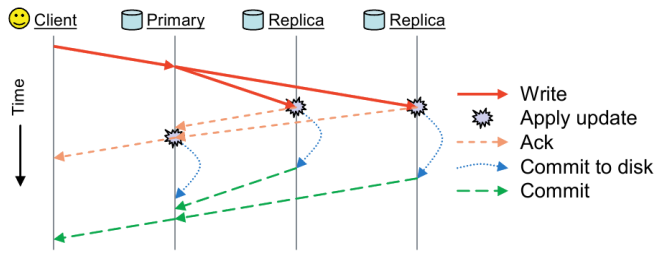
- Repliki po zapisaniu danych na dysk wysyłają do głównego OSD komunikat **commit**. Główny OSD po otrzymaniu takiego komunikatu i gdy sam zapisze dane na dysku wysyła **commit** do klienta.



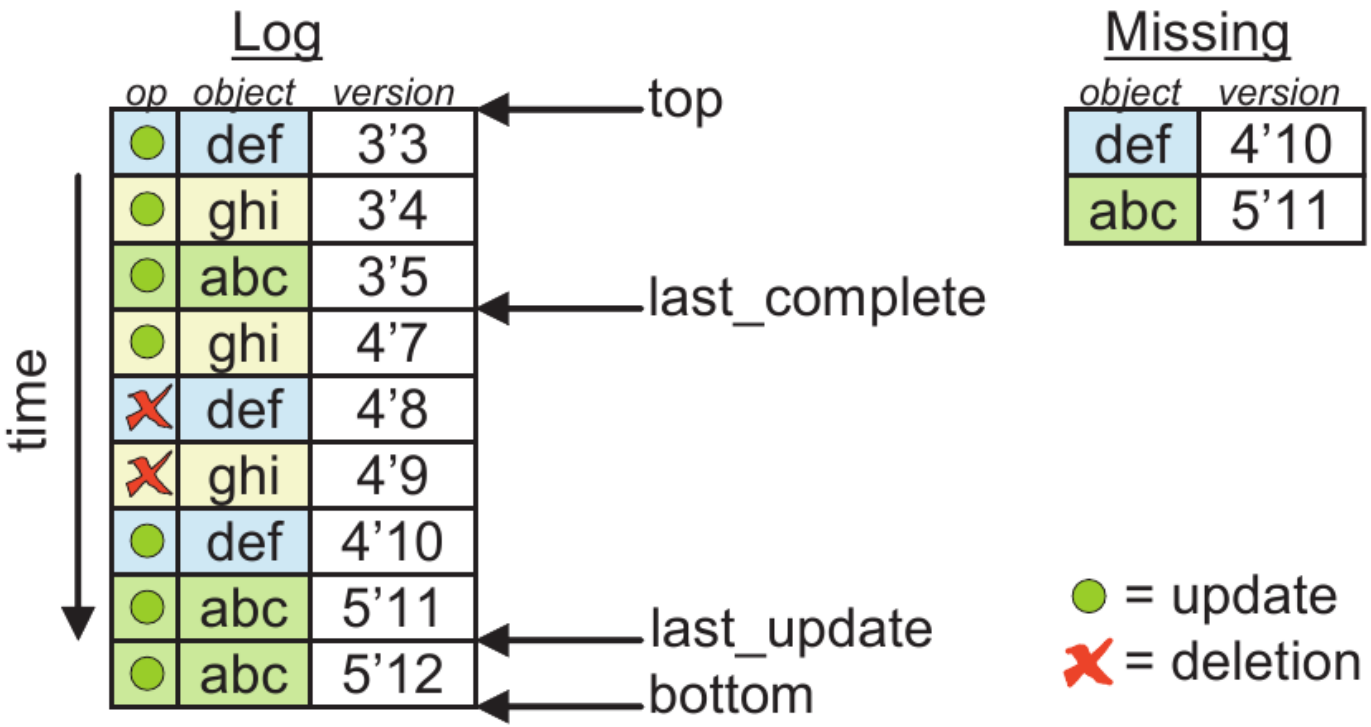
Rysunek: Zapis do klastra OSD (źródło: [2])

Zapis nowej porcji danych – ciąg dalszy:

- Repliki po zapisaniu danych na dysk wysyłają do głównego OSD komunikat **commit**. Główny OSD po otrzymaniu takiego komunikatu i gdy sam zapisze dane na dysku wysyła **commit** do klienta.
- Klient domyślnie buforuje dane do czasu otrzymania komunikatu commit.



Rysunek: Zapis do klastra OSD (źródło: [2])



Rysunek: Dziennik PG (źródło: [3])

Co się dzieje, gdy OSD dowiaduje się o zmianie w mapie klastra?

Co się dzieje, gdy OSD dowiaduje się o zmianie w mapie klastra?

- **active set** – maszyny obsługujące dany PG w aktualnej epoce (przed udostępnieniem nowej mapy).

Co się dzieje, gdy OSD dowiaduje się o zmianie w mapie klastra?

- **active set** – maszyny obsługujące dany PG w aktualnej epoce (przed udostępnieniem nowej mapy).
- OSD przegląda listę utrzymywanych lokalnie PG (wcale nie musi należeć do **active set**). Sprawdza dla których PG nastąpiła zmiana składu maszyn OSD (względem ostatniej epoki) i oznacza takie PG jako **nieaktywne** (nie obsługuje żądań klientów).

- Jeżeli dla jakiegoś zmienionego PG maszyna OSD nie jest maszyną główną, to jest wysyłany komunikat do aktualnej, nowej maszyny głównej. W komunikacie znajdują się wartości **last_update** oraz **last_complete** dla lokalnie utrzymywanego PG, a także wartość **last_epoch_started** informująca o ostatniej epoce, w której OSD należał do **active set** danego PG.

- Jeżeli dla jakiegoś zmienionego PG maszyna OSD nie jest maszyną główną, to jest wysyłany komunikat do aktualnej, nowej maszyny głównej. W komunikacie znajdują się wartości **last_update** oraz **last_complete** dla lokalnie utrzymywanego PG, a także wartość **last_epoch_started** informująca o ostatniej epoce, w której OSD należał do **active set** danego PG.
- Nowa maszyna główna tworzy zbiór **prior set**, do którego należą wszystkie maszyny, dla których **last_epoch_started** $>=$ **max(last_epoch_started)**.

- Wszystkie maszyny z **prior set** są bezpośrednio odpytywane przez główny OSD.

- Wszystkie maszyny z **prior set** są bezpośrednio odpytywane przez główny OSD.
- Główny OSD odbudowuje dziennik dla danego PG. Jeśli dziennik został skrócony (np. zawiera zmiany tylko z ostatniego tygodnia) to główny OSD może zażądać **backlog** od maszyn OSD. **Backlog** to dziennik, który zawiera spis obiektów utrzymywanych przez OSD dla danego PG.

- Wszystkie maszyny z **prior set** są bezpośrednio odpytywane przez główny OSD.
- Główny OSD odbudowuje dziennik dla danego PG. Jeśli dziennik został skrócony (np. zawiera zmiany tylko z ostatniego tygodnia) to główny OSD może zażądać **backlog** od maszyn OSD. **Backlog** to dziennik, który zawiera spis obiektów utrzymywanych przez OSD dla danego PG.
- Wszystkie maszyny z **prior set** (oprócz tych z nowego **active set**) są wzywane do aktualizacji wartości **last_epoch_started** przez nową maszynę główną. Zmodyfikowany PG jest aktywowany na wszystkich maszynach ze zbioru **active set**.

Odzyskiwanie utraconych bądź nieposiadanych obiektów w obrębie PG odbywa się za pośrednictwem głównego OSD. Prosta zasada: najpierw główny OSD musi uzyskać obiekt, dopiero potem będzie go rozsyłać do pozostałych replik. Wynika z tego, że zapytania o brakujący obiekt kierowane do głównego OSD będą blokować do czasu uzyskania obiektu. Autorzy implementując taki mechanizm chcieli uniknąć zbędnych odczytów z dysków oraz kłopotów z synchronizacją.

Twórcy Ceph-u zdecydowali się na zaimplementowanie własnego systemu pliku o nazwie EBOFS (Extent and Btree based Object File System). Nie jest to system ogólnego przeznaczenia. Czynniki, które wpłynęły na decyzję o użyciu własnego systemu plików to:

Twórcy Ceph-u zdecydowali się na zaimplementowanie własnego systemu pliku o nazwie EBOFS (Extent and Btree based Object File System). Nie jest to system ogólnego przeznaczenia. Czynniki, które wpłynęły na decyzję o użyciu własnego systemu plików to:

- Semantyka Posix jest niedostosowana do potrzeb systemu Ceph (brak grupowania operacji w transakcje).

Twórcy Ceph-u zdecydowali się na zaimplementowanie własnego systemu pliku o nazwie EBOFS (Extent and Btree based Object File System). Nie jest to system ogólnego przeznaczenia. Czynniki, które wpłynęły na decyzję o użyciu własnego systemu plików to:

- Semantyka Posix jest niedostosowana do potrzeb systemu Ceph (brak grupowania operacji w transakcje).
- Buforowanie istniejące w VFS jest obliczone na innego typu operacje oraz obciążenia.

Właściwości systemu plików EBOFS:

- Obsługa grupowania operacji w transakcje.

Właściwości systemu plików EBOFS:

- Obsługa grupowania operacji w transakcje.
- System plików działa w przestrzeni użytkownika. Wszelkie operacje na dysku są wykonywane z pomocą dyskowego urządzenia blokowego.

Właściwości systemu plików EBOFS:

- Obsługa grupowania operacji w transakcje.
- System plików działa w przestrzeni użytkownika. Wszelkie operacje na dysku są wykonywane z pomocą dyskowego urządzenia blokowego.
- Obsługa wywołania zwrotnego w przypadku, gdy dane zostały bezpiecznie zapisane na dysku.

Właściwości systemu plików EBOFS:

- Obsługa grupowania operacji w transakcje.
- System plików działa w przestrzeni użytkownika. Wszelkie operacje na dysku są wykonywane z pomocą dyskowego urządzenia blokowego.
- Obsługa wywołania zwrotnego w przypadku, gdy dane zostały bezpiecznie zapisane na dysku.
- B-drzewo jest używane do zarządzania metadanymi (na przykład do odnajdywania obiektów lub wolnego miejsca na dysku).

Właściwości systemu plików EBOFS:

- Obsługa grupowania operacji w transakcje.
- System plików działa w przestrzeni użytkownika. Wszelkie operacje na dysku są wykonywane z pomocą dyskowego urządzenia blokowego.
- Obsługa wywołania zwrotnego w przypadku, gdy dane zostały bezpiecznie zapisane na dysku.
- B-drzewo jest używane do zarządzania metadanymi (na przykład do odnajdywania obiektów lub wolnego miejsca na dysku).
- Opcjonalna obsługa dziennika.

Właściwości systemu plików EBOFS – ciąg dalszy:

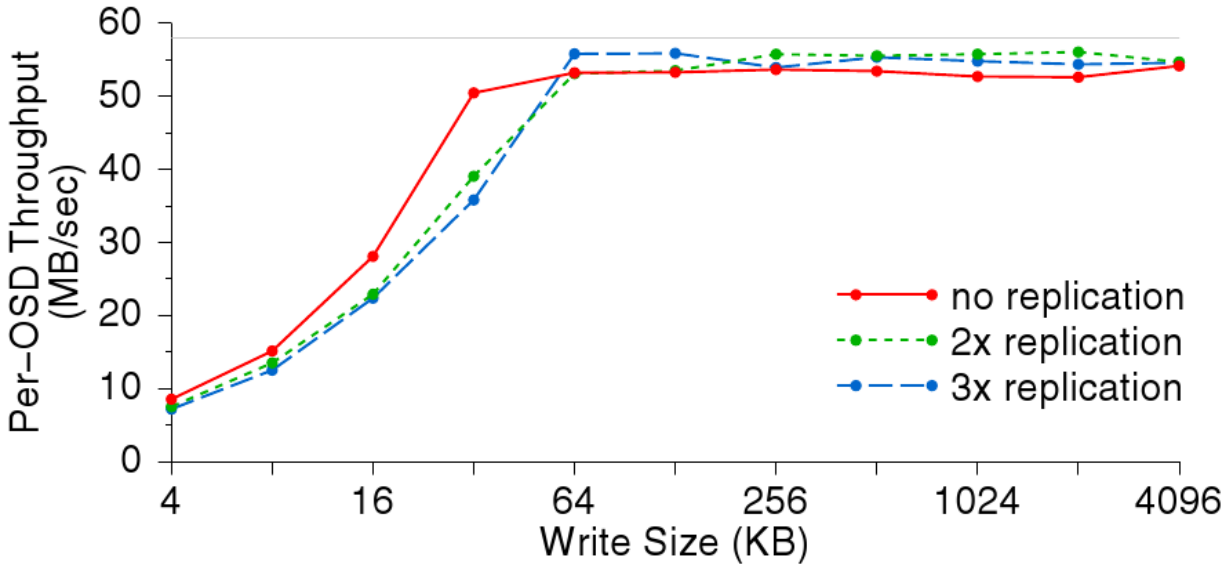
- Dane i metadane (z wyjątkiem dwóch superbloków) są zawsze zapisywane w niezarezerwowane miejsce.

Właściwości systemu plików EBOFS – ciąg dalszy:

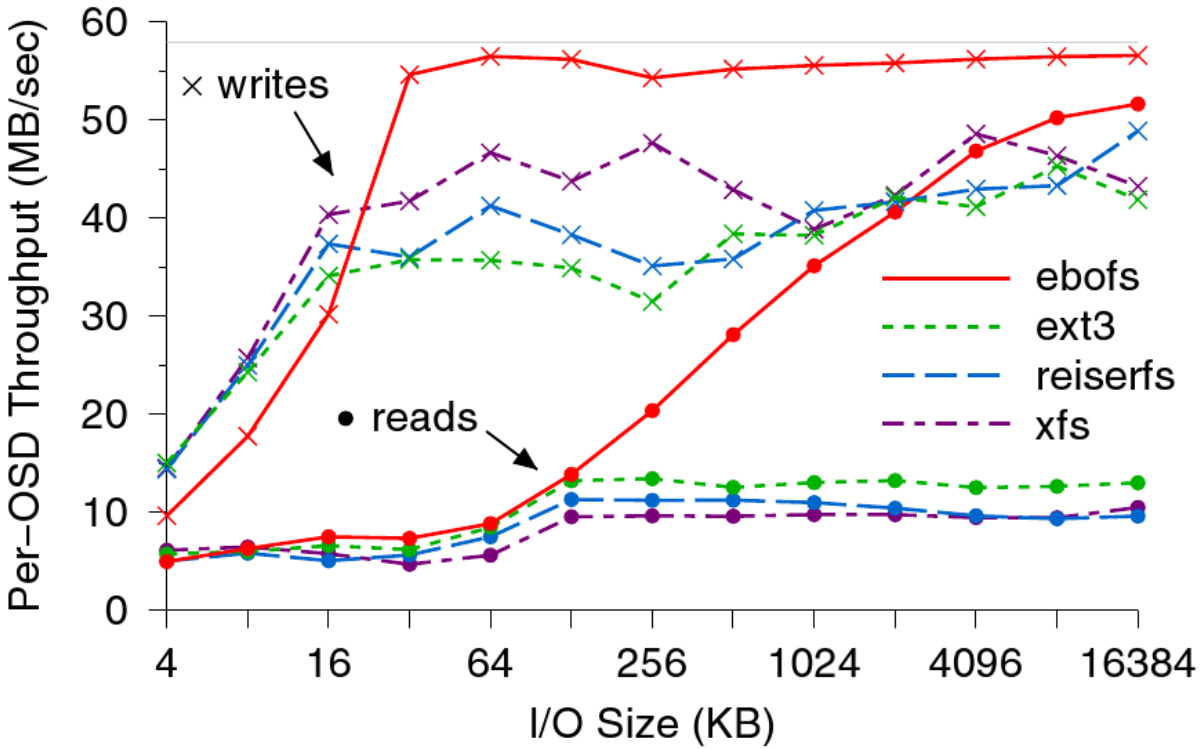
- Dane i metadane (z wyjątkiem dwóch superbloków) są zawsze zapisywane w niezarezerwowane miejsce.
- Dwa superbloki reprezentują dwie epoki danych i metadanych, które cyklicznie się zastępują. Podczas zmiany epoki następuje dokończenie buforowanych operacji i ustawienie wskaźnika na nowsze metadane w jednym z superbloków.

Właściwości systemu plików EBOFS – ciąg dalszy:

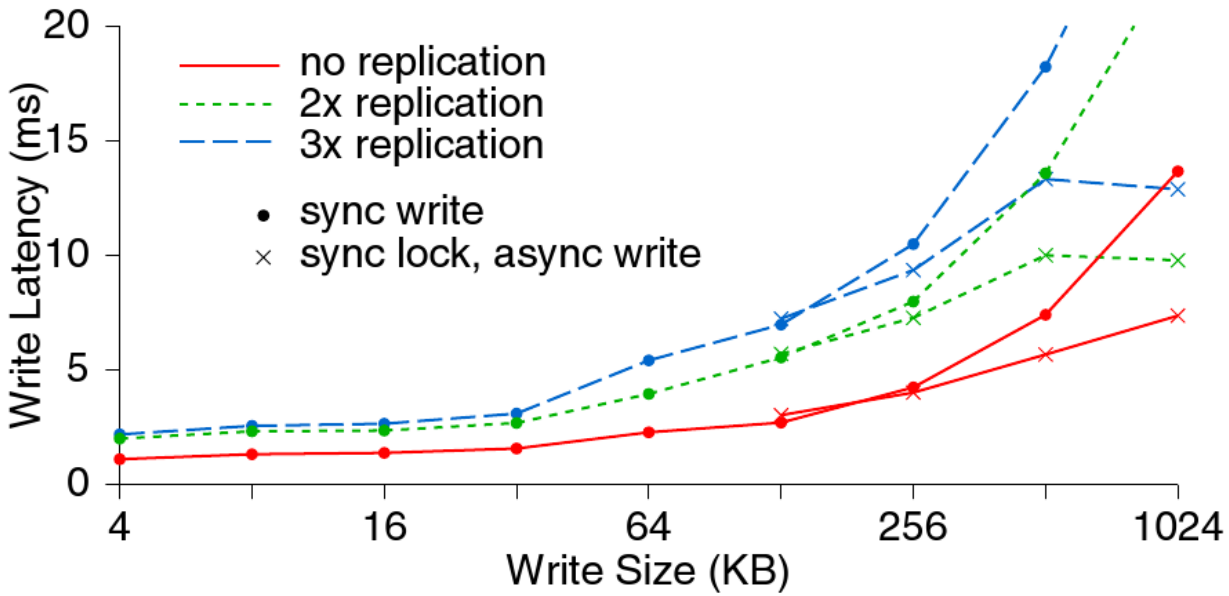
- Dane i metadane (z wyjątkiem dwóch superbloków) są zawsze zapisywane w niezarezerwowane miejsce.
- Dwa superbloki reprezentują dwie epoki danych i metadanych, które cyklicznie się zastępują. Podczas zmiany epoki następuje dokończenie buforowanych operacji i ustawienie wskaźnika na nowsze metadane w jednym z superbloków.
- Agresywne buforowanie danych (sprawdzanie tuż przed wysłaniem danych na dysk czy nowe operacje nie zdezaktualizowały danych).



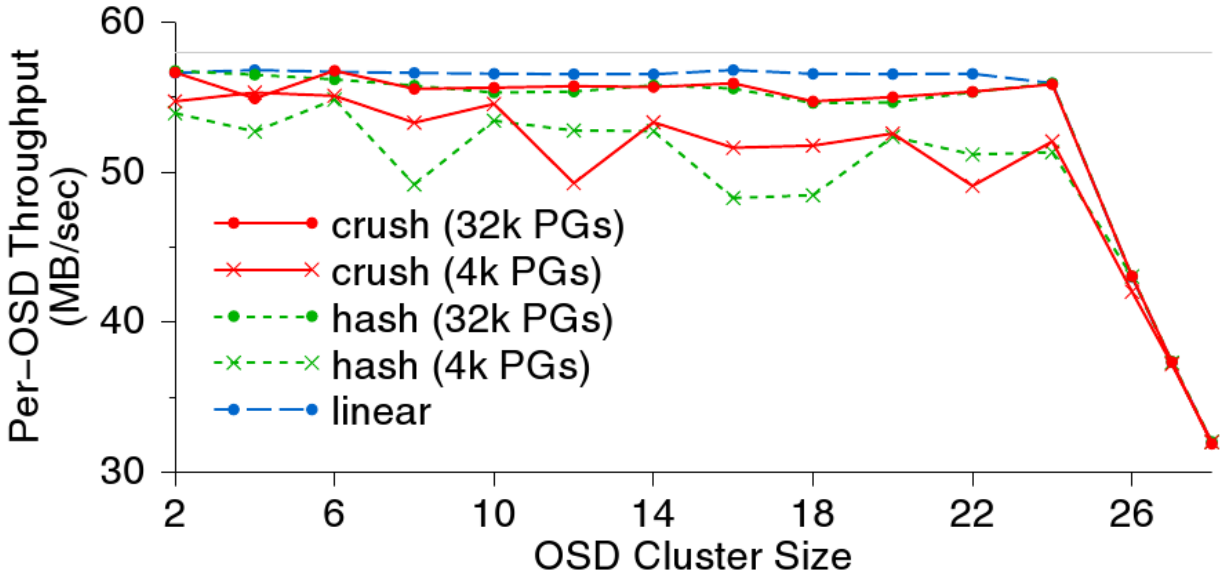
Rysunek: Przepustowość zapisów na pojedynczej maszynie OSD (źródło: [2])



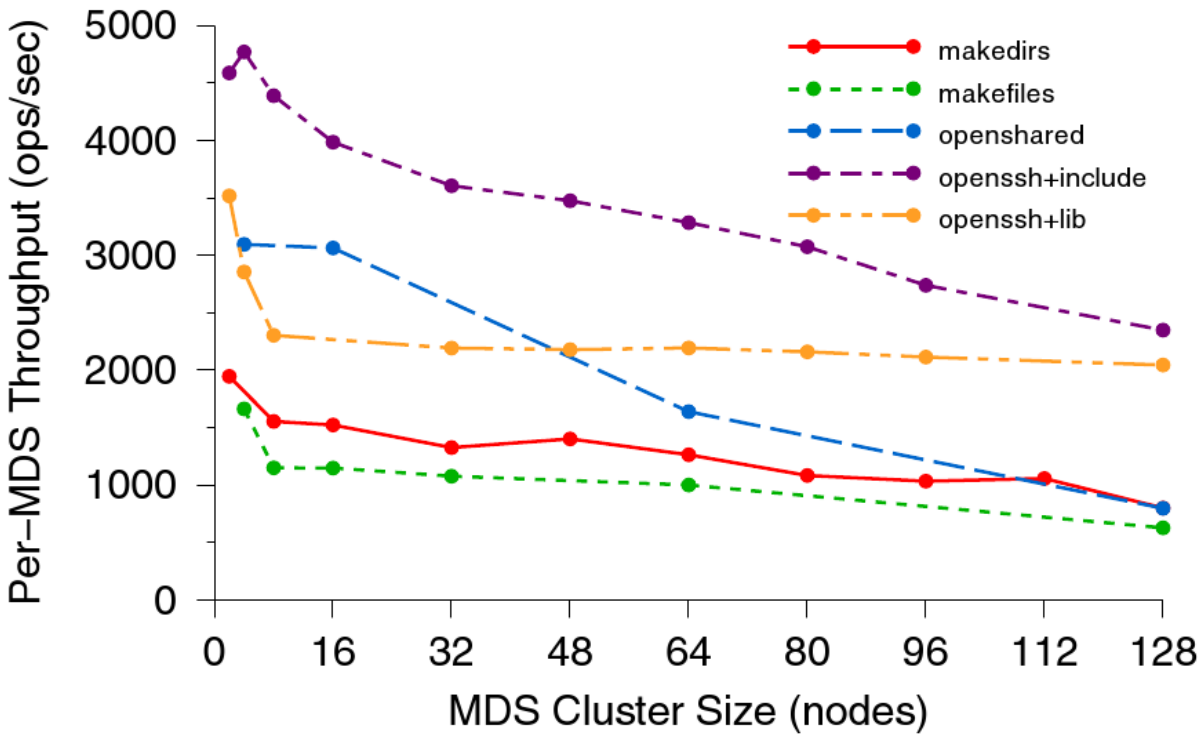
Rysunek: Wydajność systemu plików EBOFS (źródło: [2])



Rysunek: Opóźnienia zapisu (źródło: [2])



Rysunek: Przepustowość na pojedynczej maszynie OSD w zależności od wielkości klastra (źródło: [2])



Rysunek: Przepustowość na pojedynczej maszynie MDS w zależności od wielkości klastra (źródło: [2])

- http://ceph.newdream.net/
- Sage Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, Carlos Maltzahn, Ceph: A Scalable, High-Performance Distributed File System
- Sage A. Weil. Ceph: Reliable, Scalable, and High-Performance Distributed Storage. Ph.D. thesis, University of California, Santa Cruz, December, 2007
- Sage Weil, Kristal Pollack, Scott A. Brandt, Ethan L. Miller, Dynamic Metadata Management for Petabyte-Scale File Systems