System plików JFS

Mateusz Zakrzewski

18 stycznia 2004

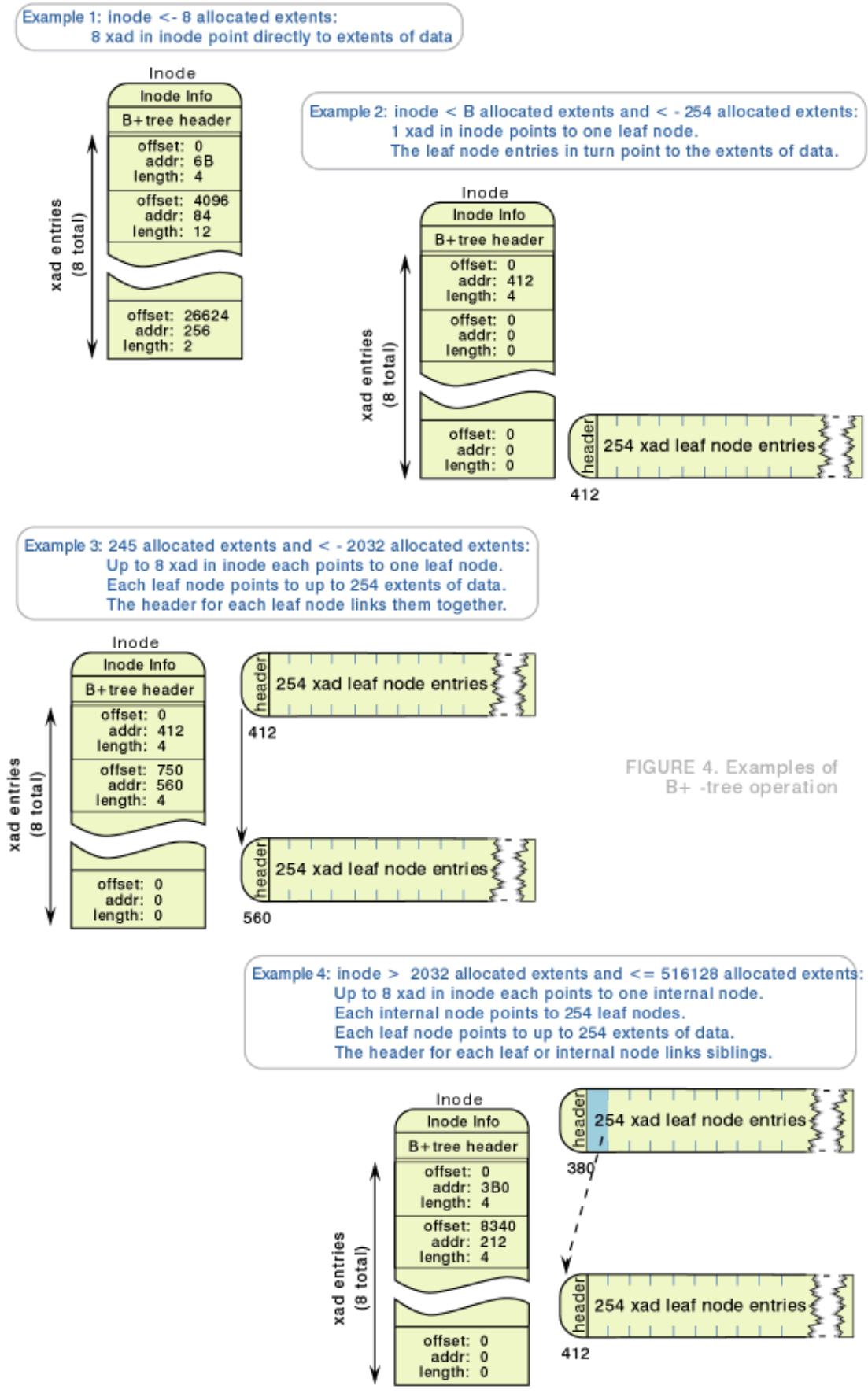
1 Najważniejsze informacje.

- JFS to skrót od Journalled File System.
- Stworzony przez firmę IBM w 1990 roku.
- Jest niezawodny, dzięki księgowaniu.
- Zamiast partycji są *agregaty* i *zestawy plików*.
- Jest skalowalny tzn. można łatwo zmieniać rozmiar agregatu.

Pod pojęciem *partycja* można rozumieć coś co można montować lub część dysku na której znajdują się dane zorganizowane w sposób niezależny od danych znajdujących się w innych częściach dysku. W przypadku innych systemów plików np. w ReiserFs to jest to samo tzn. dane, które tworzą pewien montowalny zbiór plików i katalogów są zapisane w pewnej części dysku i aby np. stworzyć nowy plik nie trzeba korzystać z danych zapisanych w innych częściach dysku. W przypadku JFS to nie jest to samo, ponieważ w jednym *agregacie*, czyli w pewnej części dysku mogą być zapisane dane dotyczące wielu różnych *zestawów plików*, czyli montowalnych zbiorów plików i katalogów.

Ta różnica pomiędzy JFS a np. ReiserFs jest widoczna również z punktu widzenia użytkownika tzn. użytkownik może mieć dwie partycje z systemem plików ReiserFs. Kopiuje dane z zewnętrznego nośnika danych na pierwszą partycję. Jest możliwe, że nie będzie mógł skopiować wszystkich danych z powodu braku wolnego miejsca na pierwszej partycji, mając dużo wolnego miejsca na drugiej partycji. W systemie plików JFS, użytkownik może mieć dwa zestawy plików należące do jednego agregatu. Wtedy taka sytuacja nie będzie możliwa, ponieważ dane opisujące, które części dysku są zajęte, a które wolne są wspólne dla wszystkich zestawów plików należących do tego samego agregatu.

2 Sposób przechowywania informacji o pliku.



Dane o pliku są przechowywane w i-węźle. I-węzeł jest korzeniem B+drzewa. W przypadku małych plików dane są trzymane bezpośrednio w i-węźle. W przypadku większych plików poszczególne części

pliku są trzymane w ekstentach. W i-węźle trzymane są wskaźniki do ekstentów lub, jeśli jest to konieczne tzn. gdy plik składa się z wielu ekstentów, wskaźniki do struktur zawierających wskaźniki do ekstentów lub wskaźniki do struktur zawierających wskaźniki do struktur zawierających wskaźniki do ekstentów itd...

Można zauważyć, że w ten sposób można trzymać nie tylko plik, ale również np. tablicę wskaźników do dowolnych obiektów. B+drzewo w którym na najniższym poziomie są ekstenty nadaje się do przechowywania dowolnego obiektu, którego różne części znajdują się w różnych miejscach dysku i którego część możemy znaleźć podając offset w tym obiekcie.

Struktura opisująca ekstent:

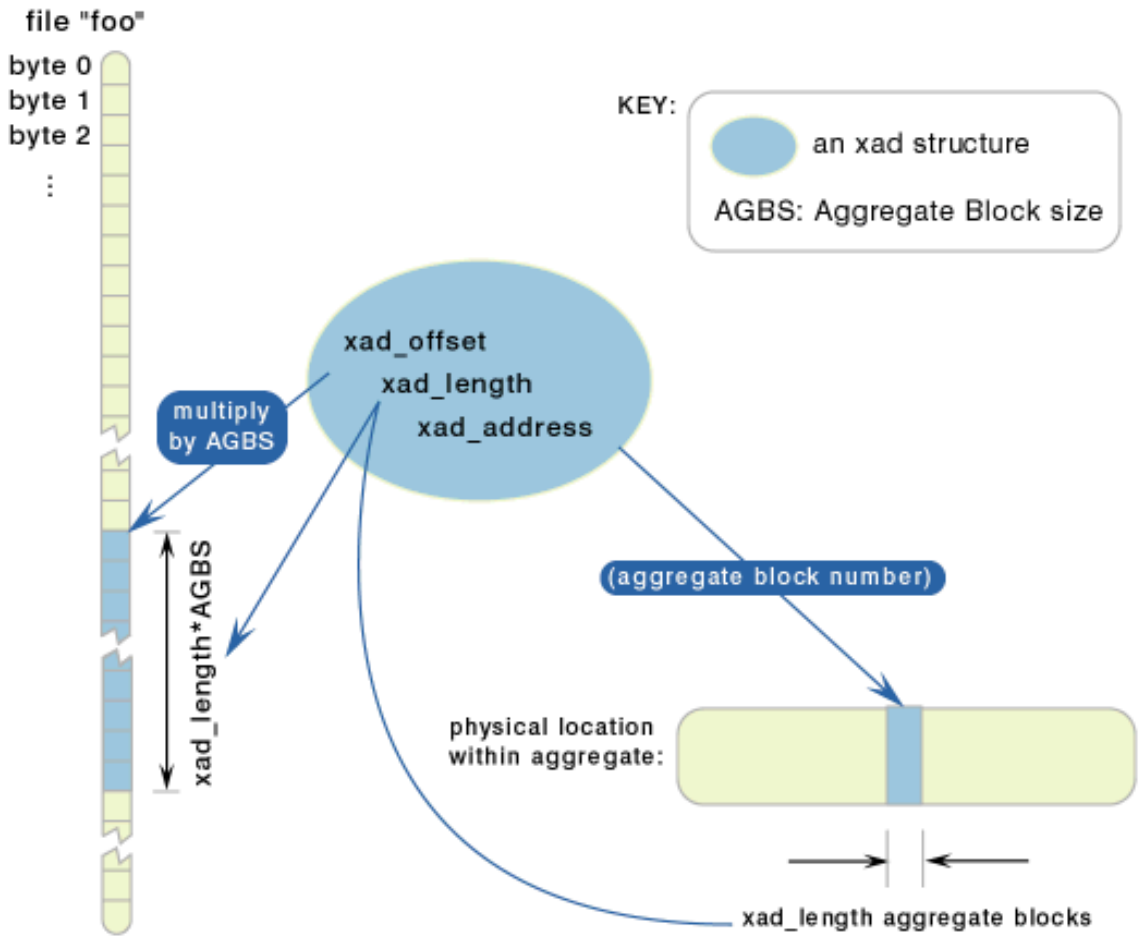


FIGURE 3. xad describes two "ranges"

- Pole address, to adres pierwszego bloku ekstentu.

- Pole length, to długość ekstentu wyrażona w ilości bloków.

- Pole offset opisuje, którą część pliku zawiera dany ekstent tzn. zaczynającą się od jakiego offsetu, wyrażoną w ilości bloków agregatu. Pole offset jest kluczem w B+drzewie.

Pliki mogą być przechowywane w różny sposób:

1041377-bajtowy plik, przechowywany w jednym ekstencie:

```
offset      0
length      1017
address     xxxxx
```

Ten sam plik w trzech kawałkach:

```
offset      0
length      495
address     xxxxx

offset      495
length      22
address     yyyyy

offset      517
length      500
address     zzzzz
```

Plik powstały w wyniku wykonania operacji:

```
fd = creat("nowy plik", ...);
write (fd, "ab", 2);
lseek (fd, 1041374, 0);
write (fd, "cde" , 3);

offset      0
length      1
address     xxxxx

offset      1016
length      1
address     yyyyy
```

Jak widać, powyższy plik zajmuje fizycznie mniej miejsca na dysku, niż wynosi jego logiczny rozmiar. Jeżeli wykonana zostanie próba odczytania bajtów ze środka pliku, to zwrócone zostaną bajty zerowe (tak powinno być), pomimo tego, że nie są one fizycznie przechowywane na dysku.

Plik mający 16GB, zalokowany w sposób ciągły:

```
offset      0
length      16777215
address     12345

offset      16777215
length      1
address     16789560
```

Powyższy plik musi być opisany przez dwie struktury, pomimo tego, że jest zaalokowany w sposób ciągły, ponieważ na pole length nie można przypisać wartości większej niż $2^{24} - 1 = 16777215$.

3 Ogólny układ danych w agregacie.

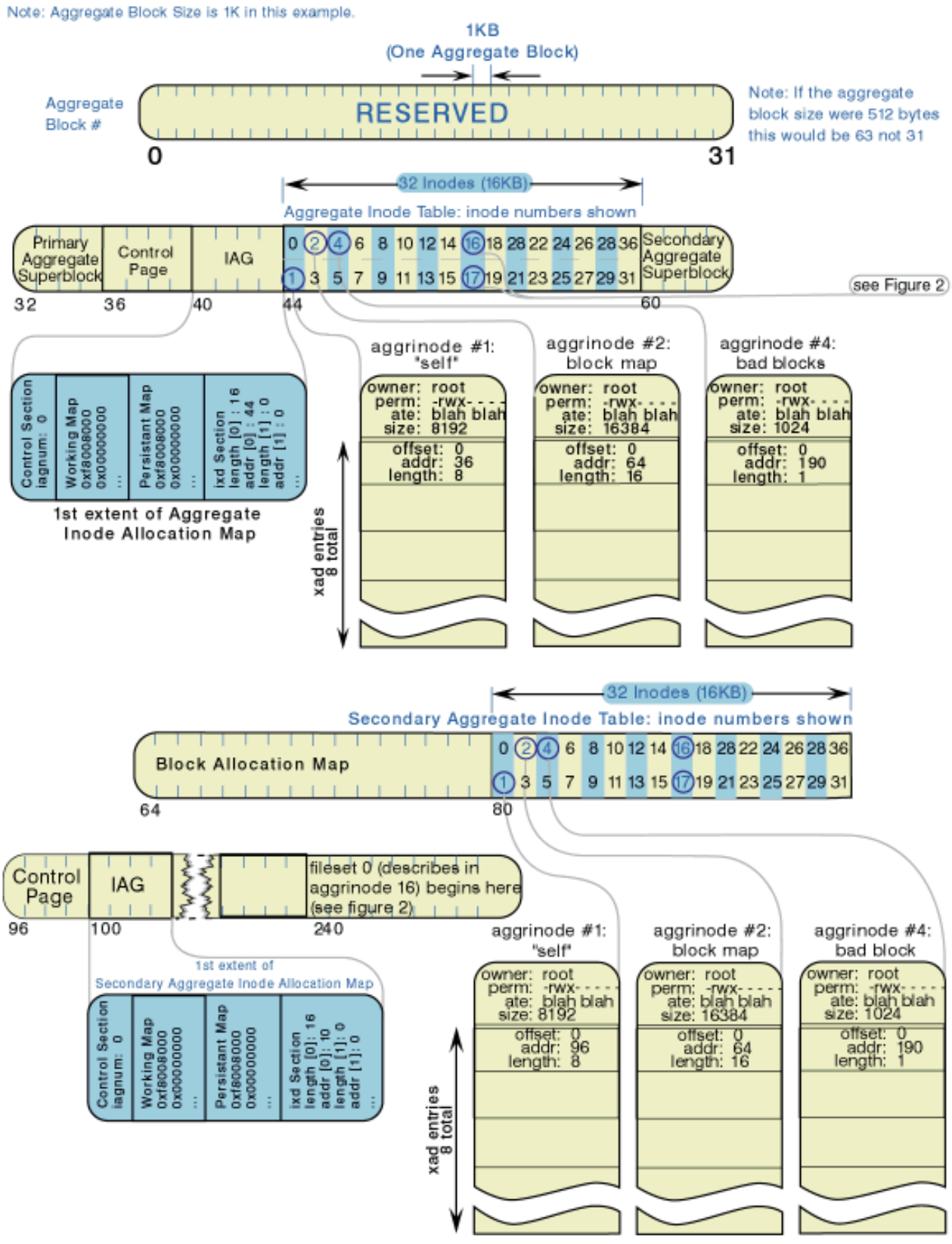


FIGURE1. The Big Picture: An aggregate with two filesets

3.1 Zawartość agregatu:

- Dwa superbloki. Zawierają one informacje dotyczące całego agregatu takie jak: rozmiar agregatu, rozmiar grup alokacji, rozmiar bloku agregatu itd. Rozmiar bloku, to najmniejsza przydzialana lub zwalniana ilość miejsca na dysku. Grupa alokacji, to struktura opisująca pewną część agregatu, który jest podzielony na grupy alokacji. Jeżeli chcemy zaalokować wiele bloków leżących blisko siebie, to alokujemy bloki należące do tej samej grupy alokacji.

  Superbloki znajdują się zawsze w tych samych ustalonych pozycjach. Drugi superblok jest kopią pierwszego. Jest on używany, gdy pierwszy superblok jest uszkodzony.

- Tablica i-węzłów agregatu. W tej tablicy znajdują się i-węzły opisujące dane dotyczące całego agregatu.

- Druga tablica i-węzłów agregatu. Jest kopią pierwszej. Jest potrzebna, na wypadek, gdyby pierwsza została uszkodzona.

- Mapa alokacji bloków. Mapa alokacji bloków opisuje, które bloki są zajęte, a które wolne.

- Dziennik (nie pokazany na rysunku), czyli miejsce na dysku potrzebne do stosowania księgowania.

3.2 Zawartość tablicy i-węzłów agregatu:

- I-węzeł opisujący samą tablicę i-węzłów agregatu. Można go znaleźć, ponieważ znajduje się on w ustalonym miejscu.

- I-węzeł opisujący położenie mapy alokacji bloków.

- I-węzeł opisujący położenie dziennika.

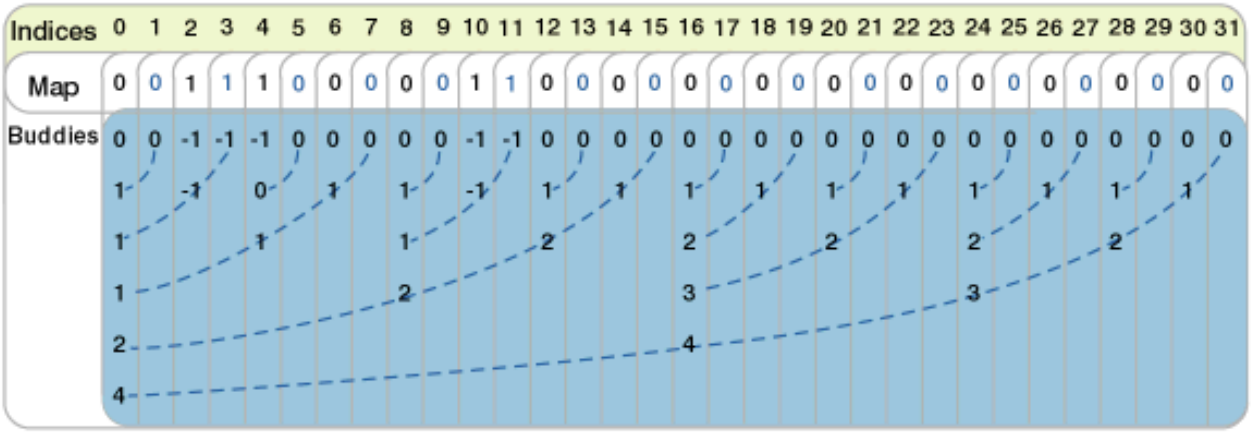
- I-węzeł opisujący położenie uszkodzonych bloków agregatu.

- I-węzły opisujące położenie struktur opisujących poszczególne zestawy plików.

Można zauważyć, że wiele struktur (np. mapa alokacji bloków lub mapa i-węzłów agregatu) jest opisanych tak samo, jak są opisane pliki tzn. poprzez i-węzły. Wynika z tego, że można łatwo zmieniać ich rozmiar, ponieważ można to robić w taki sam sposób w jaki się zmienia rozmiar pliku np. pisząc coś do niego. Dzięki takiej architekturze systemu plików JFS, można łatwo zmieniać rozmiar agregatu.

4 Sposób szukania dużego spójnego fragmentu wolnej pamięci.

FIGURE 7. Binary Buddy of a word of the bitmap



Trzeba umieć szybko znajdować spójne fragmenty dysku złożone z wolnych bloków, aby szybko alokować nowe ekstenty. Można wykorzystać do tego algorytm bliźniaków. Jest on pokazany na rysunku.

- W polu map, 0 oznacza blok wolny, a 1 oznacza blok zajęty.
- W polu buddies, jeżeli na n-tym poziomie (licząc od góry), na m-tej pozycji jest liczba k i $k \neq -1$, i $n \neq 1$, to znaczy, że rozmiar największego spójnego fragmentu wolnej pamięci w obszarze od pozycji m, do pozycji $m + 2^{n-1} - 1$ jest większy lub równy 2^k. Liczba -1 oznacza, że odpowiedni blok jest w całości zajęty.
- Ten algorytm nie jest dokładny, ale jest szybki.

4.1 Struktura mapy alokacji bloków.

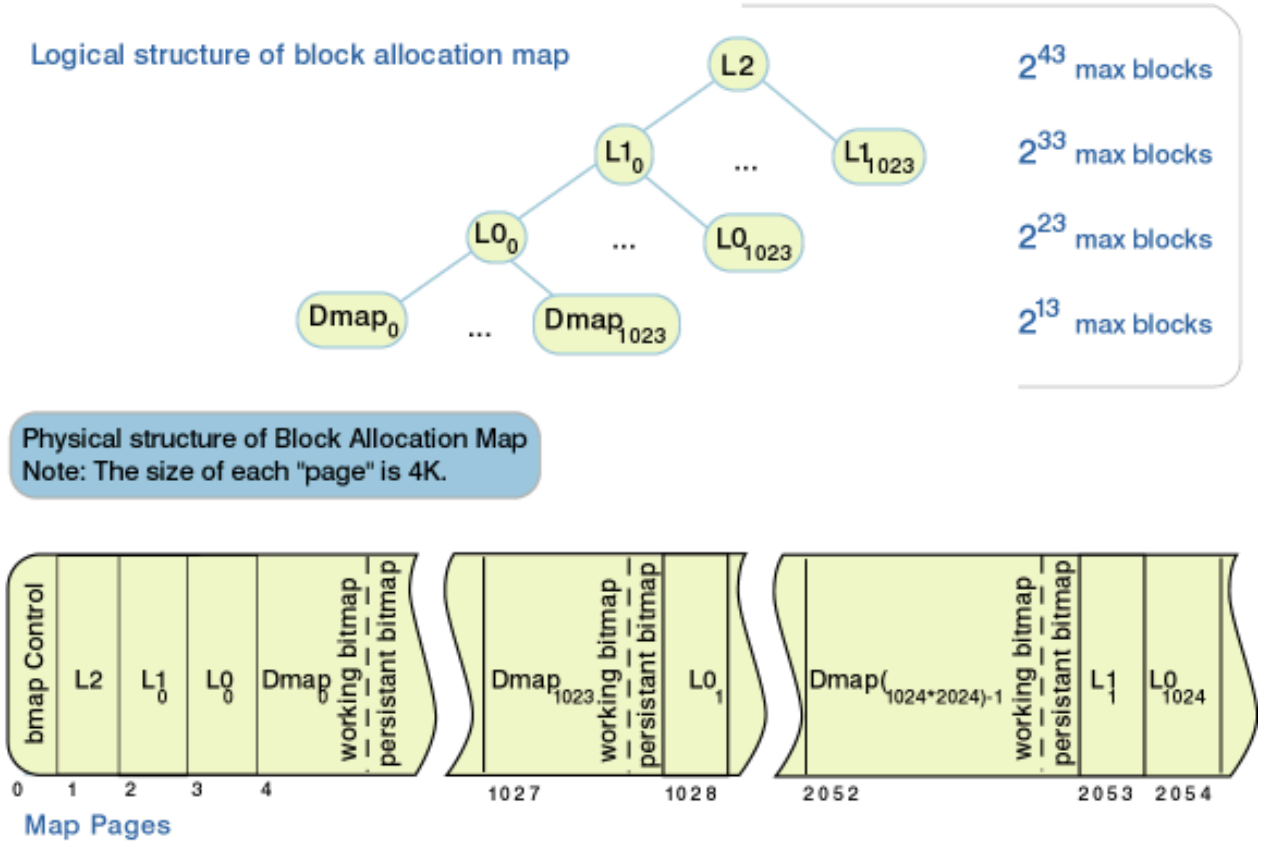


FIGURE 5. Block Allocation Map

Na rysunku widać strukturę mapy alokacji bloków. Mapa alokacji bloków to drzewo, które jest reprezentowane poprzez dane zapisane w tablicy w taki sposób, że gdybyśmy przechodzili to drzewo w porządku pre-order i wypisywali zawartość kolejnych węzłów, to otrzymalibyśmy zawartość tej tablicy. W strukturze dmap zapisane są informacje o tym, który blok jest zajęty, a który wolny, oraz informacje o spójnych fragmentach wolnego miejsca. W strukturach na wyższych poziomach zapisane są informacje o większych spójnych fragmentach (otrzymane przy pomocy tego samego algorytmu).

Struktura dmap składa się z *mapy trwałej* i *mapy roboczej*. Mapa trwała opisuje zajętość bloków, która powinna zostać odtworzona w razie awarii. Jeżeli alokujemy nowy blok, to najpierw aktualizujemy mapę roboczą. Jeżeli zwalniamy blok, to najpierw aktualizujemy mapę trwałą.

4.2 Struktura dmap.

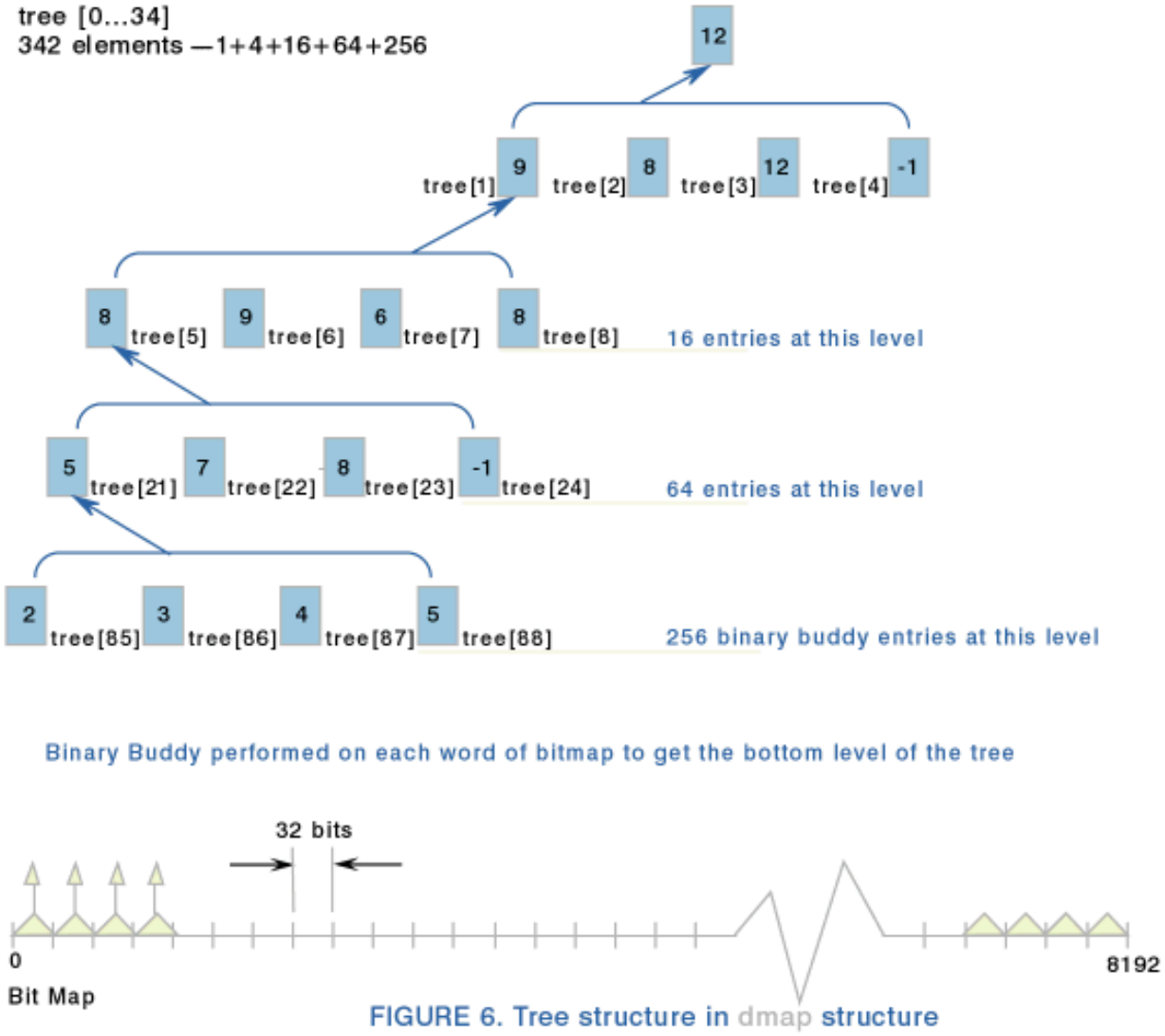


FIGURE 6. Tree structure in dmap structure

Na rysunku widać dokładnie strukturę dmap, a właściwie strukturę struktury dmap. Struktura ta zawiera, poza mapą zajętości bloków, również informacje otrzymane poprzez algorytm bliźniaków, w taki sposób, jaki został przedstawiony na rysunku na stronie 7 tzn. najpierw na podstawie każdych 32 bitów obliczana jest liczba opisująca najdłuższy spójny fragment wolnego miejsca, a następnie na podstawie każdych czterech takich liczb na niższym poziomie obliczana jest liczba na wyższym poziomie.

5 Więcej informacji:

- http://www-106.ibm.com/developerworks/library/l-jfs.html
- http://www-106.ibm.com/developerworks/library/l-jfslayout/