

FILESYSTEM TESTING

su piattaforma

DELL PowerEdge 1950

Versione: 0.2
Data: 12.02.2007
Riservatezza: N/A
Numero di pagine: 30

AUTORE: Gabriele Cicala
REVISIONE: N/A
APPROVAZIONE: N/A

Executive summary

Il presente documento costituisce la versione 0.2 dei risultati conseguiti una volta determinata una metodologia per il testing del filesystem.

Sommario

<u>Specifiche tecniche</u>3
<u>Macchina</u>3
<u>Software</u>3
<u>Hardware</u>3
<u>Hard disc (hdparm)</u>5
<u>Filesystem testati</u>5
<u>EXT2</u>5
<u>EXT3</u>5
<u>JFS</u>5
<u>REISERFS V3</u>5
<u>REISERFS V4</u>6
<u>XFS</u>6
<u>RAMFS</u>6
<u>Metodologia utilizzata</u>6
<u>IOzone</u>6
<u>Bonnie++</u>7
<u>Risultati IOzone</u>10
<u>EXT2</u>11
<u>EXT3</u>13
<u>JFS</u>15
<u>REISERFS V3</u>17
<u>XFS</u>19
<u>RAMFS con dimensione 1G</u>21
<u>Risultati Bonnie++</u>23
<u>n=1</u>23
<u>n=1024</u>26
<u>Conclusioni</u>29

Specifiche tecniche

Macchina

DELL POWEREDGE 1950

Software

KERNEL: 2.6.18-3-686

Hardware

Processore

```
globo:~# cat /proc/cpuinfo
processor      : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 15
model name    : Intel(R) Xeon(R) CPU      5130 @ 2.00GHz
stepping       : 6
cpu MHz       : 1995.104
cache size    : 4096 KB
physical id   : 0
siblings       : 2
core id       : 0
cpu cores     : 2
fdiv_bug      : no
hlt_bug       : no
f00f_bug      : no
coma_bug      : no
fpu          : yes
fpu_exception : yes
cpuid level  : 10
wp           : yes
flags         : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe nx lm constant_tsc pni monitor ds_cpl vmx tm2
cx16 xtpr lahf_lm
bogomips     : 3992.97

processor      : 1
vendor_id     : GenuineIntel
cpu family    : 6
```

```
model      : 15
model name   : Intel(R) Xeon(R) CPU      5130 @ 2.00GHz
stepping    : 6
cpu MHz     : 1995.104
cache size   : 4096 KB
physical id  : 0
siblings     : 2
core id      : 1
cpu cores    : 2
fdiv_bug     : no
hlt_bug      : no
f00f_bug     : no
coma_bug     : no
fpu          : yes
fpu_exception: yes
cpuid level  : 10
wp           : yes
flags        : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe nx lm constant_tsc pni monitor ds_cpl vmx tm2
cx16 xptr lahf_lm
bogomips    : 3990.10
```

Memoria

globo:~# cat /proc/meminfo

```
MemTotal: 2076696 kB
MemFree: 1938440 kB
Buffers: 3816 kB
Cached: 49744 kB
SwapCached: 0 kB
Active: 23972 kB
Inactive: 32484 kB
HighTotal: 1179296 kB
HighFree: 1120852 kB
LowTotal: 897400 kB
LowFree: 817588 kB
SwapTotal: 4883720 kB
SwapFree: 4883720 kB
Dirty: 0 kB
Writeback: 0 kB
AnonPages: 2872 kB
Mapped: 3372 kB
```

```
Slab:           6484 kB
PageTables:     240 kB
NFS_Unstable:   0 kB
Bounce:         0 kB
CommitLimit:   5922068 kB
Committed_AS:  10036 kB
VmallocTotal:  114680 kB
VmallocUsed:   7932 kB
VmallocChunk:  106552 kB
```

Hard disc (hdparm)

Hdparm è un'interfaccia a linea di comando che permette di impostare/ottenere i parametri di device ATA/IDE usando le ioctl e supportati da Linux.

Nel caso della globo i parametri di *cache read* e *device read*:

```
hdparm -Tt /dev/sda
```

danno i seguenti risultati

Device: /dev/sda:

```
Timing cached reads: 6978 MB in 2.00 seconds = 3492.46 MB/sec
Timing buffered disk reads: 174 MB in 3.00 seconds = 57.97 MB/sec
```

N. B.: I due hard disk sono configurati in RAID1.

Filesystem testati

La partizione sotto test sulla macchina globo è la /dev/sda3

EXT2

```
mkfs.ext2 /dev/sda3
```

EXT3

```
mkfs.ext3 /dev/sda3
```

JFS

```
mkfs.jfs -q /dev/sda3
```

REISERFS V3

```
mkreiserfs -q /dev/sda3
```

REISERFS V4

`mkfs.reiser4 -y /dev/sda3`

N. B.: Non testato. Sarebbe necessario patchare e/o compilare il kernel Linux.

XFS

`mkfs.xfs -y /dev/sda3`

RAMFS

`mount -t ramfs /dev/ram## /mnt/ramdisc`

Metodologia utilizzata

IOzone

IOzone [1] è un tool di benchmarking dove il target è il testing del filesystem. Il benchmark genera e misura i vari tipi di operazione che possono essere eseguiti su di un file. Il benchmark testa le performance di Input/Output per le seguenti operazioni:

Read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read/write, pread/pwrite variants, aio_read, aio_write, mmap,

Le operazioni di nostro interesse sono generalmente la *read*, la *write*, la *rand-write* e la *rand-read*. L'analisi più approfondita può essere utile per ulteriori sviluppi del progetto e, in tal caso, si rimanda al file XLS contenente tutti i grafici.

L'output è completamente in forma numerica ma grazie a un foglio excel con macro fornita da Don Capps (Iozone.Support@gmail.com) si ottengono i grafici visualizzati di seguito.

Per quanto riguarda una maggiore delucidazione su ogni test si rimanda al documento di descrizione incluso nel package.

Versione del software in uso

Version: 3.281

Sintassi usata

`iozone -Rab iozone_test_“FSType”.xls -g 3G`

Dove con FSType si intende il nome del FS sotto test.

I parametri indicano la creazione di un file excel (-R), generazione automatica del test (-a), indicazione del file di output in formato excel (-b) e dimensione massima del file di test (-g).

Il filesystem RamFS è stato testato con dimensione massima di file pari a 1G invece che 3G, per ovvie limitazioni a causa della RAM pari a 2GB.

N.B.: Per lanciare il test è necessario copiare il binario all'interno della partizione che si vuole testare.

Bonnie++

Bonnie++ [2] è un altro benchmark il cui scopo è il testing delle performance del filesystem e dell'hard disk. Il tool testa le performance creando vari file ed emulando quindi un funzionamento tipico di un database o di altri software che durante il loro funzionamento debbono pesantemente agire sull'hard disk.

Versione del software in uso

Version: 1.03

Sintassi usata

Verranno fatti due tipi di test in cui il parametro *n* varrà rispettivamente 1 e 1024. Tale parametro indica il numero di file per il test di creazione dei file. Il metro di misura è per multipli di 1024 file.

```
bonnie++ -u root -d /data/ -s 4G -n 1 > /test/bonnie_"FSType"_n_1.out
```

```
bonnie++ -u root -d /data/ -s 4G -n 1024 > /test/bonnie_"FSType"_n_1024.out
```

Dove con FSType si intende il nome del FS sotto test.

I parametri indicano la directory di test (-d), la dimensione del file per le operazioni di I/O performance (-g) e utente con cui lanciare il test (-u).

Dettagli del test (dal sito del produttore

<http://www.coker.com.au/bonnie++/readme.html>):

- The file IO tests are:

1. **Sequential Output**

1. Per-Character. The file is written using the putc() stdio macro. The loop that does the writing should be small enough to fit into any reasonable I-cache. The CPU overhead here is that required to do the stdio code plus the OS file space allocation.
2. Block. The file is created using write(2). The CPU overhead should be just the OS file space allocation.

3. Rewrite. Each BUFSIZ of the file is read with read(2), dirtied, and rewritten with write(2), requiring an lseek(2). Since no space allocation is done, and the I/O is well-localized, this should test the effectiveness of the filesystem cache and the speed of data transfer.
2. **Sequential Input**
1. Per-Character. The file is read using the getc() stdio macro. Once again, the inner loop is small. This should exercise only stdio and sequential input.
 2. Block. The file is read using read(2). This should be a very pure test of sequential input performance.
3. **Random Seek** This test runs SeekProcCount processes (default 3) in parallel, doing a total of 8000 lseek()s to locations in the file specified by random() in bsd systems, drand48() on sysV systems. In each case, the block is read with read(2). In 10% of cases, it is dirtied and written back with write(2). The idea behind the SeekProcCount processes is to make sure there's always a seek queued up. AXIOM: For any unix filesystem, the effective number of lseek(2) calls per second declines asymptotically to near 30, once the effect of caching is defeated. One thing to note about this is that the number of disks in a RAID set increases the number of seeks. For read using RAID-1 (mirroring) will double the number of seeks. For write using RAID-0 will multiply the number of writes by the number of disks in the RAID-0 set (provided that enough seek processes exist). The size of the file has a strong nonlinear effect on the results of this test. Many Unix systems that have the memory available will make aggressive efforts to cache the whole thing, and report random I/O rates in the thousands per second, which is ridiculous. As an extreme example, an IBM RISC 6000 with 64 Mb of memory reported 3,722 per second on a 50 Mb file. Some have argued that bypassing the cache is artificial since the cache is just doing what it's designed to. True, but in any application that requires rapid random access to file(s) significantly larger than main memory which is running on a system which is doing significant other work, the caches will inevitably max out. There is a hard limit hiding behind the cache which has been observed by the author to be of significant importance in many situations - what we are trying to do here is measure that number.
- The file creation tests use file names with 7 digits numbers and a random number (from 0 to 12) of random alpha-numeric characters. For the sequential tests the random characters in the file name follow the number. For the random tests the random characters are first. The sequential tests involve creating the files in numeric order, then stat()ing them in readdir() order (IE the order they are stored in the directory which is very likely to be the same order as which they were

created), and deleting them in the same order. For the random tests we create the files in an order that will appear random to the file system (the last 7 characters are in numeric order on the files). Then we stat() random files (NB this will return very good results on file systems with sorted directories because not every file will be stat()ed and the cache will be more effective). After that we delete all the files in random order. If a maximum size greater than 0 is specified then when each file is created it will have a random amount of data written to it. Then when the file is stat()ed it's data will be read.

Verranno dapprima presentati i risultati conseguiti con l'utilizzo del tool IOzone e successivamente quelli ottenuti utilizzando Bonnie++.

Risultati IOzone

I valori che verranno presi in considerazione sono le operazioni di *write*, *read*, *rand-write* e *rand-read* per eventuali approfondimenti circa le altre caratteristiche si rimanda al foglio excel di interesse.

EXT2

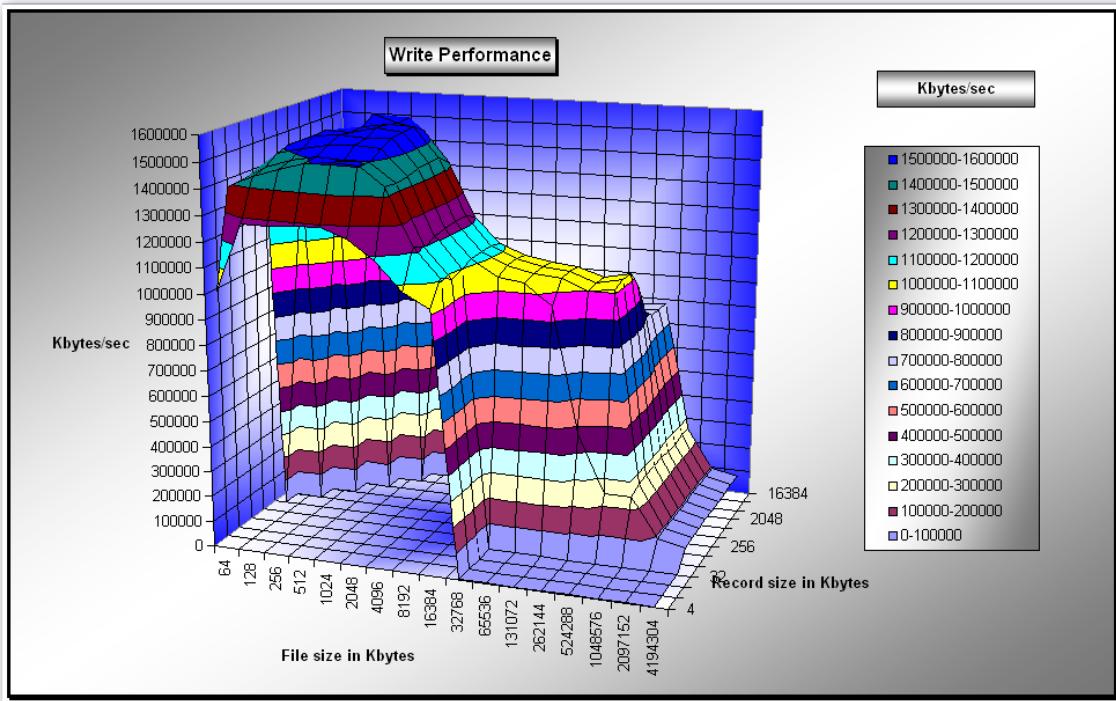


Figure 1

Per la sua caratteristica a sella si ottengono discrete performance nell'intervallo tra 128 e 4096 Kbytes.

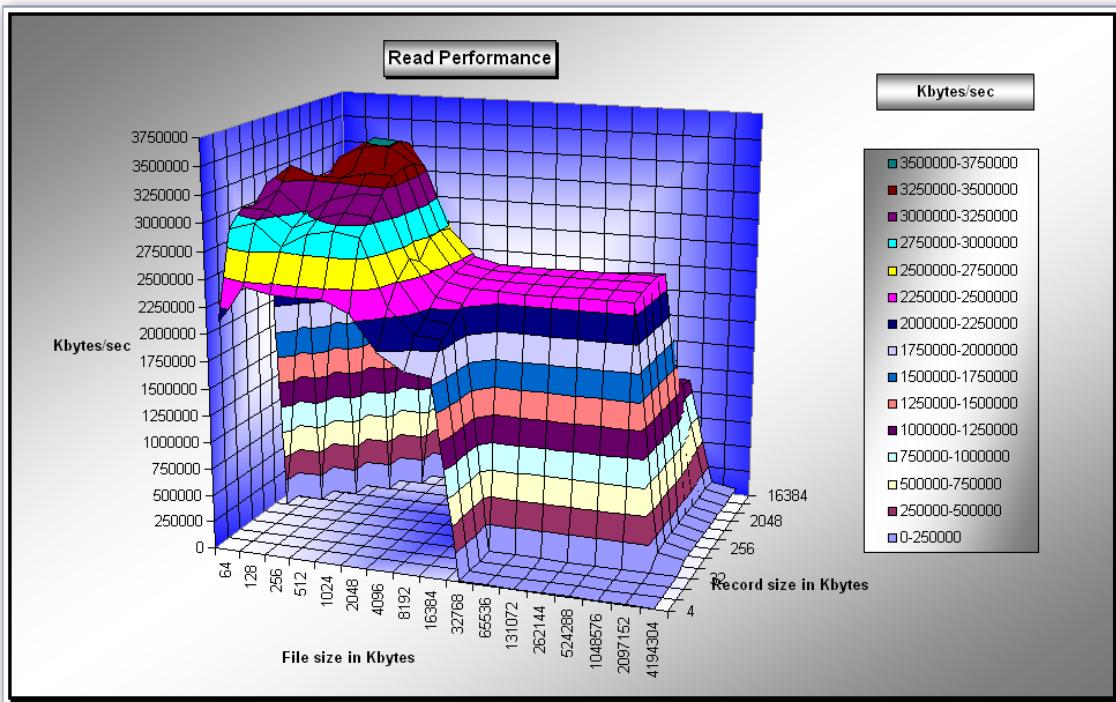


Figure 2

Idem anche in questo caso anche se la velocità di lettura risulta più alta.

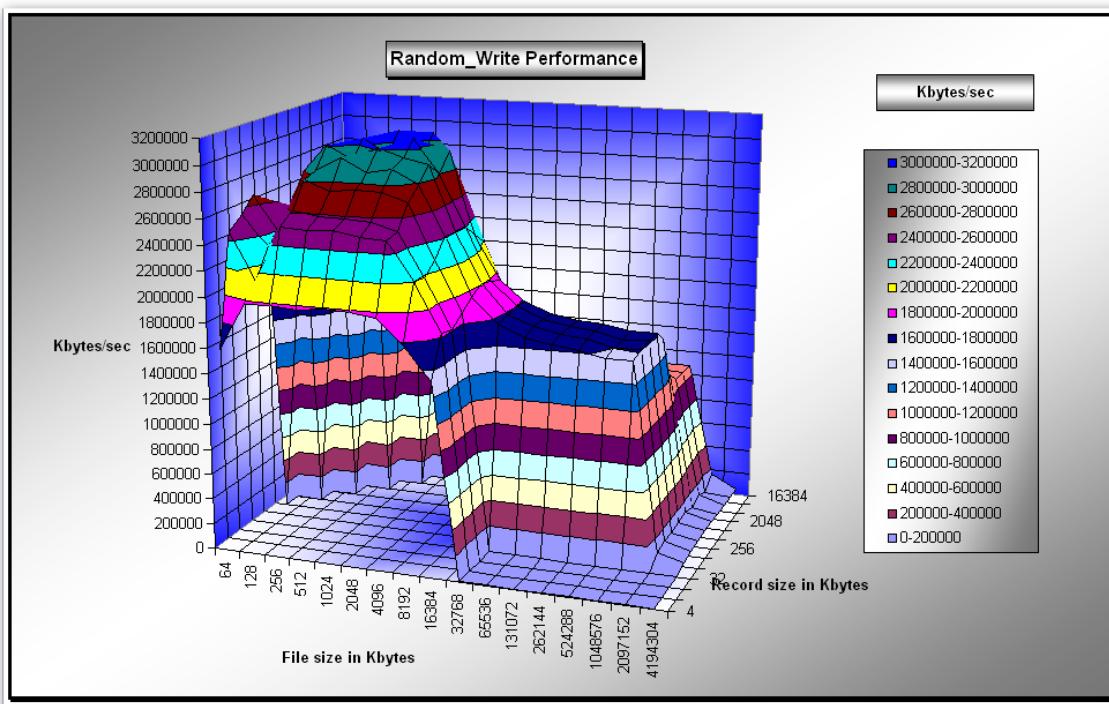


Figure 3

Nella *random-write* si ottiene un miglior profilo nell'intervallo tra 256 e 4096 Kbytes.

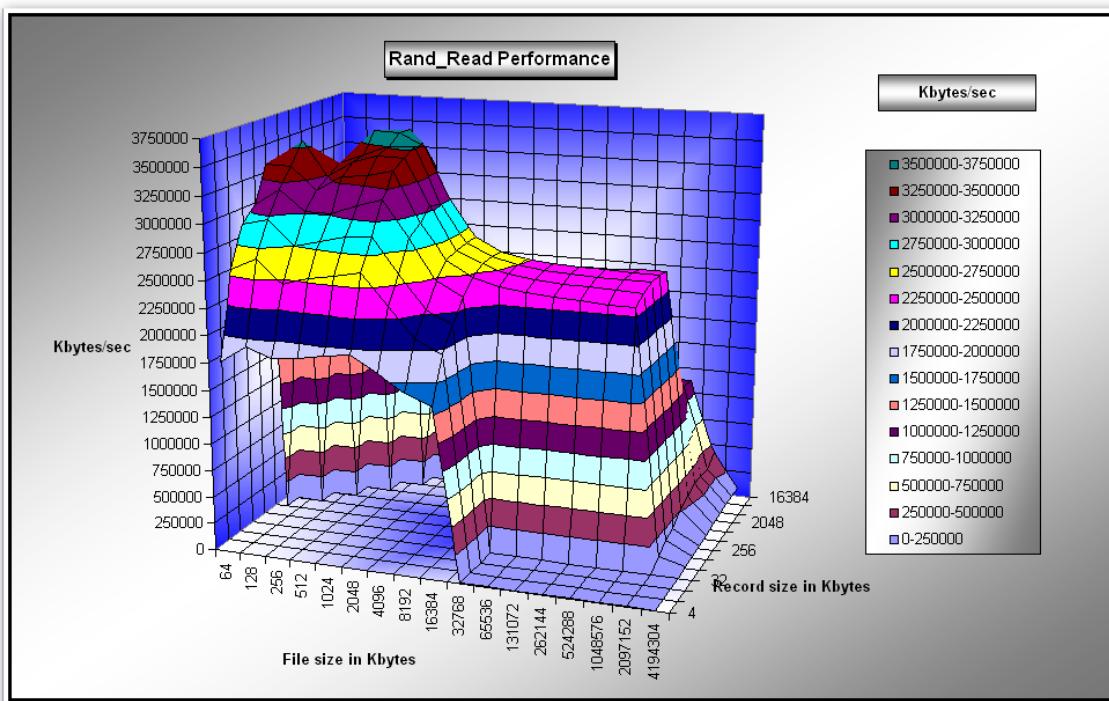


Figure 4

Anche nella *random-read* si ha un andamento particolare nell'intervallo tra 64 e 8192 Kbytes ma in generale si vede un andamento costante nel range tra 2250000 e 2500000 Kbytes/sec.

EXT3

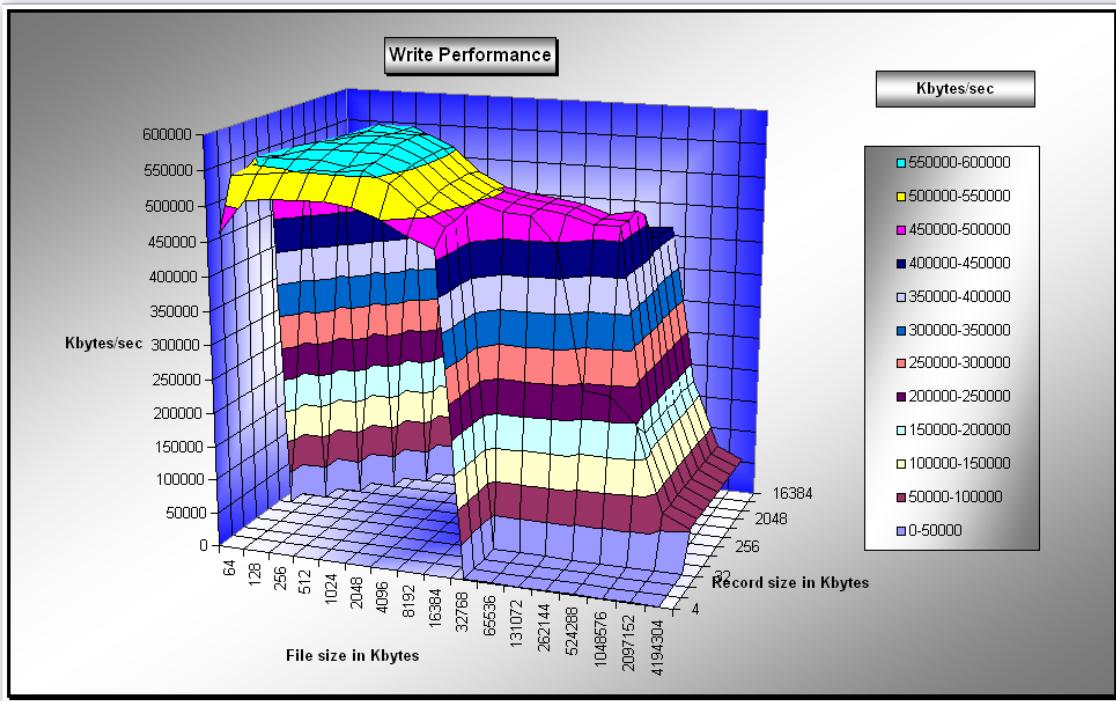


Figure 5

Nella ext3 la *write* è notevolmente bassa rispetto alla precedente (a parità di condizioni).

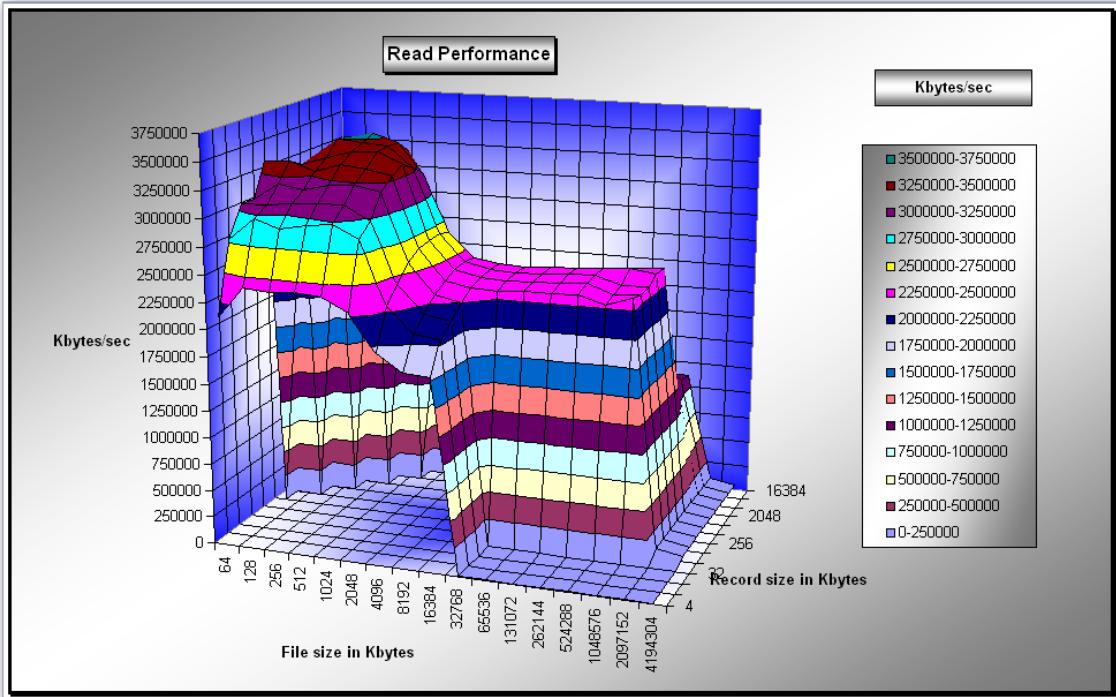


Figure 6

Nella *read*, a parte l'andamento a sella, si ha sostanzialmente una buona velocità di lettura tra 2250000 e 2500000 Kbytes/sec.

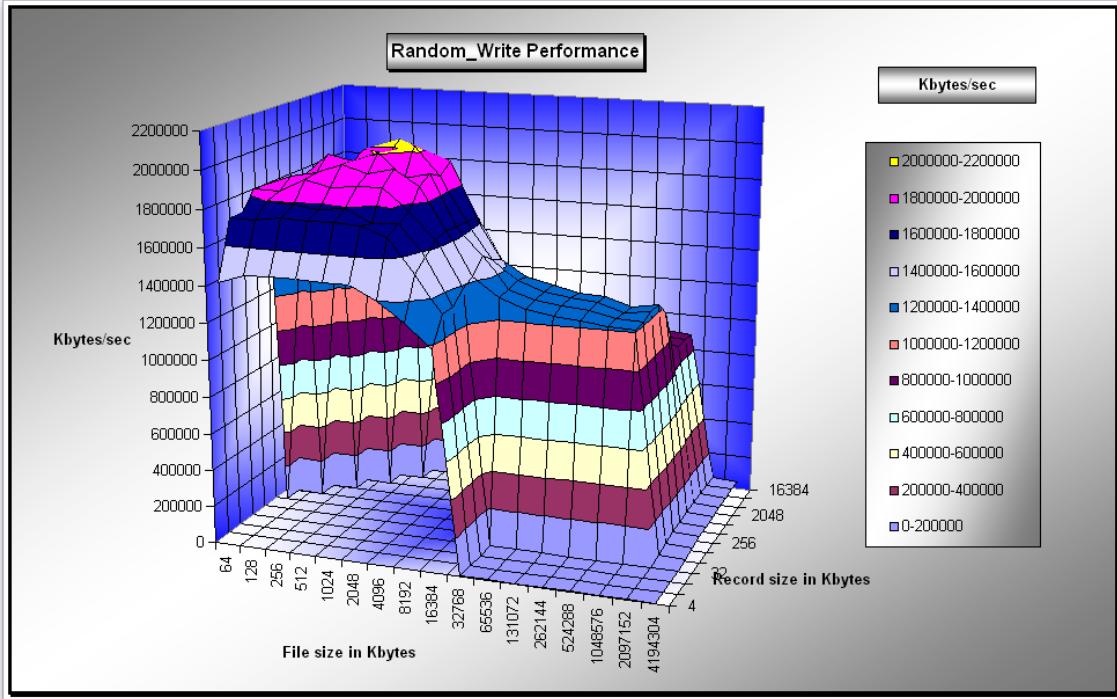


Figure 7

Rispetto alla *write* nella *random-write* si ottengono migliori prestazioni ma in ogni caso peggiori della ext2.

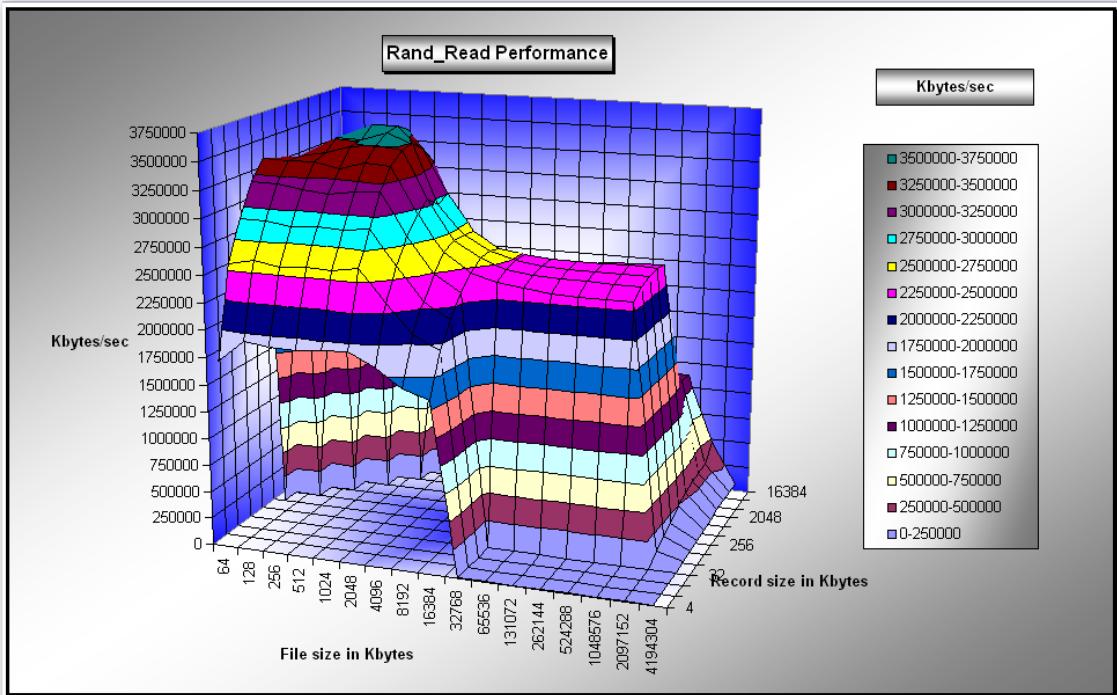


Figure 8

Anche nella *random-read* si ha un costante livello nell'intervallo tra 2250000 e 2500000 Kbytes/sec.

JFS

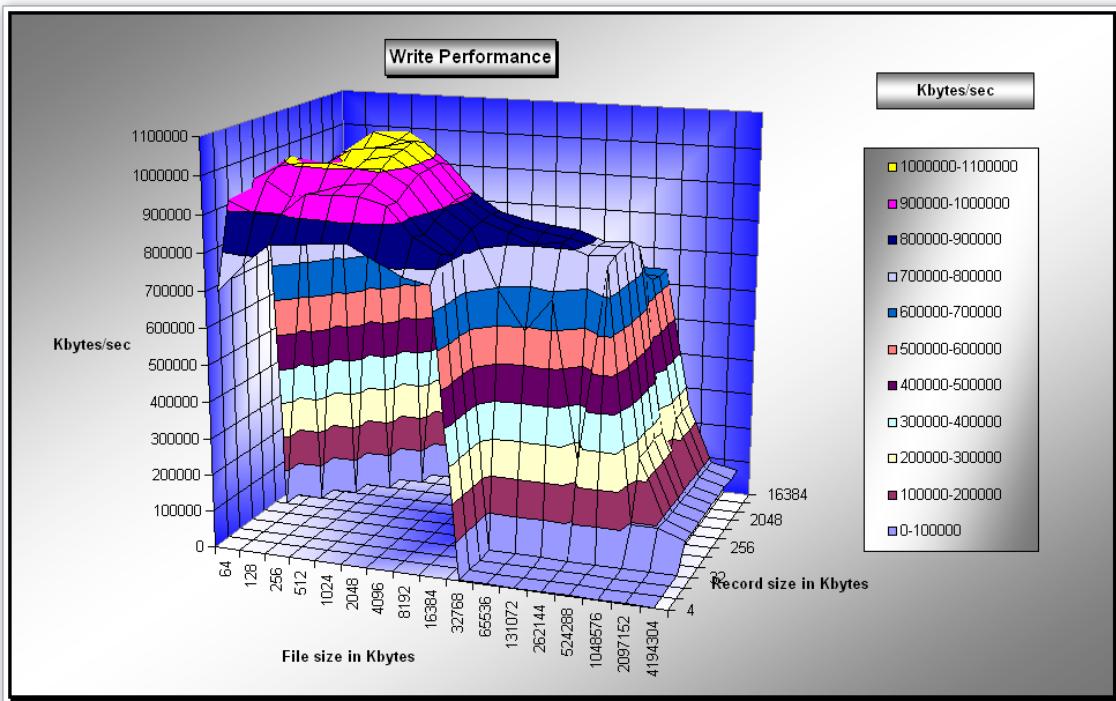


Figure 9

Nella JFS ci si alza in qualità del throughput disponibile ma in ogni caso si ha un andamento a sella entro le dimensioni di 8192 Kbytes.

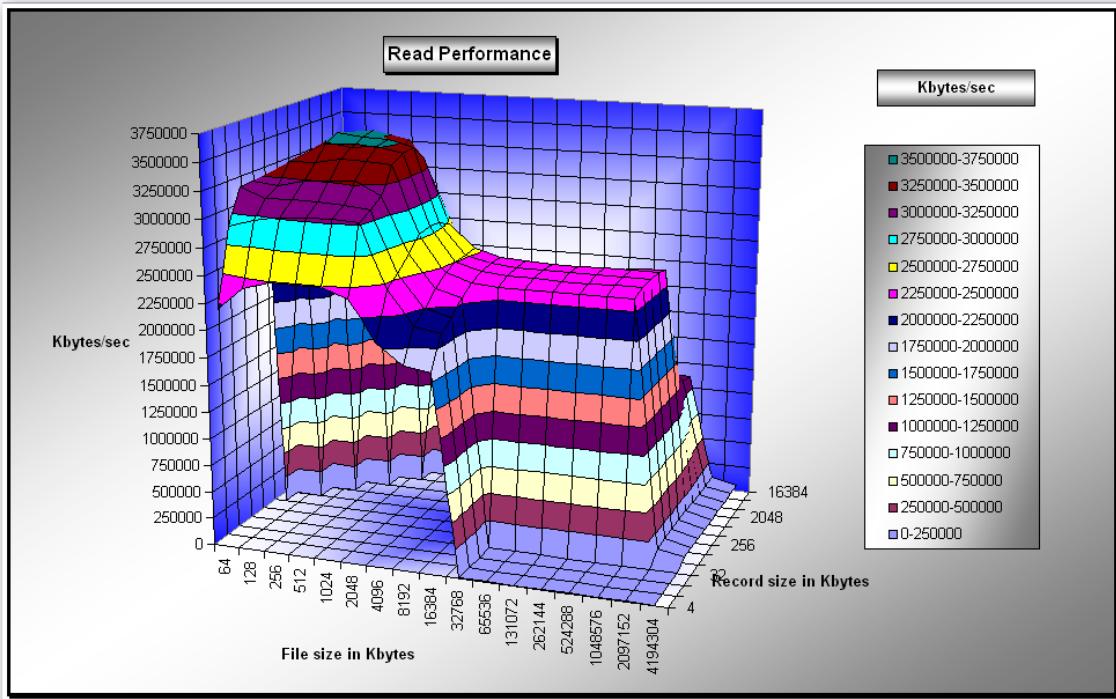


Figure 10

Anche nella *read* si ha un andamento a sella ma in ogni caso si evince un profilo discreto anche al di fuori di questo range di valori.

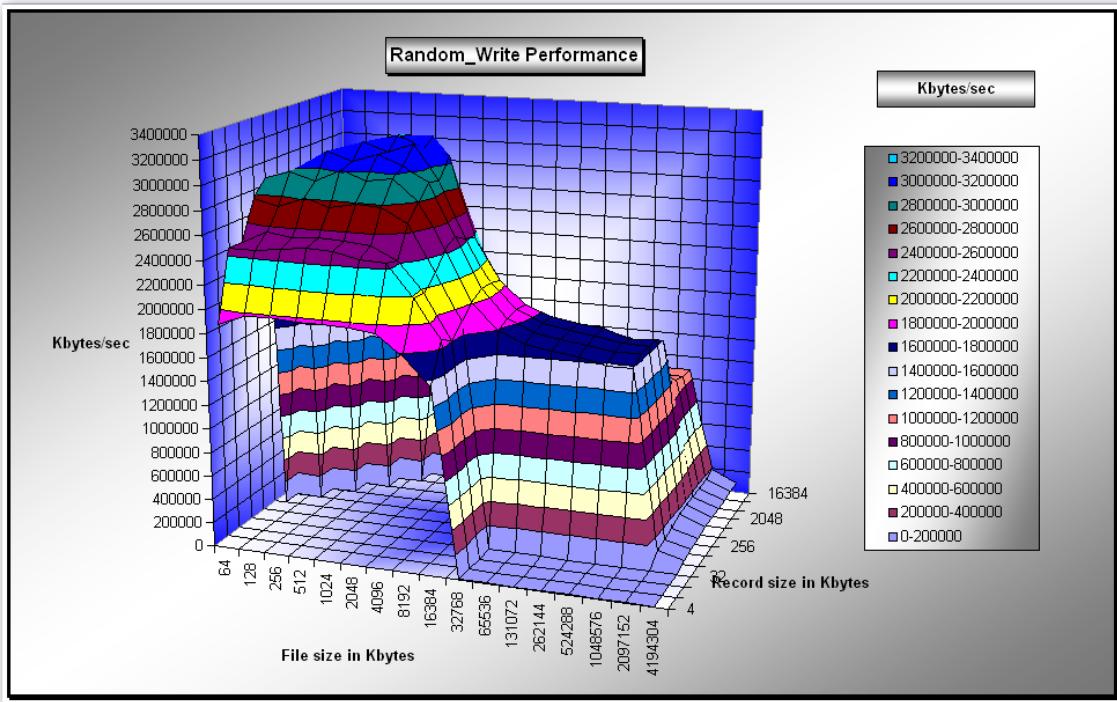


Figure 11

Nella *random-write* si ha andamento a sella entro 4096 Kbytes che dà un throughput 2000000 e 2200000 Kbytes/sec. Al di fuori di questa sella si ha un notevole deprezzamento.

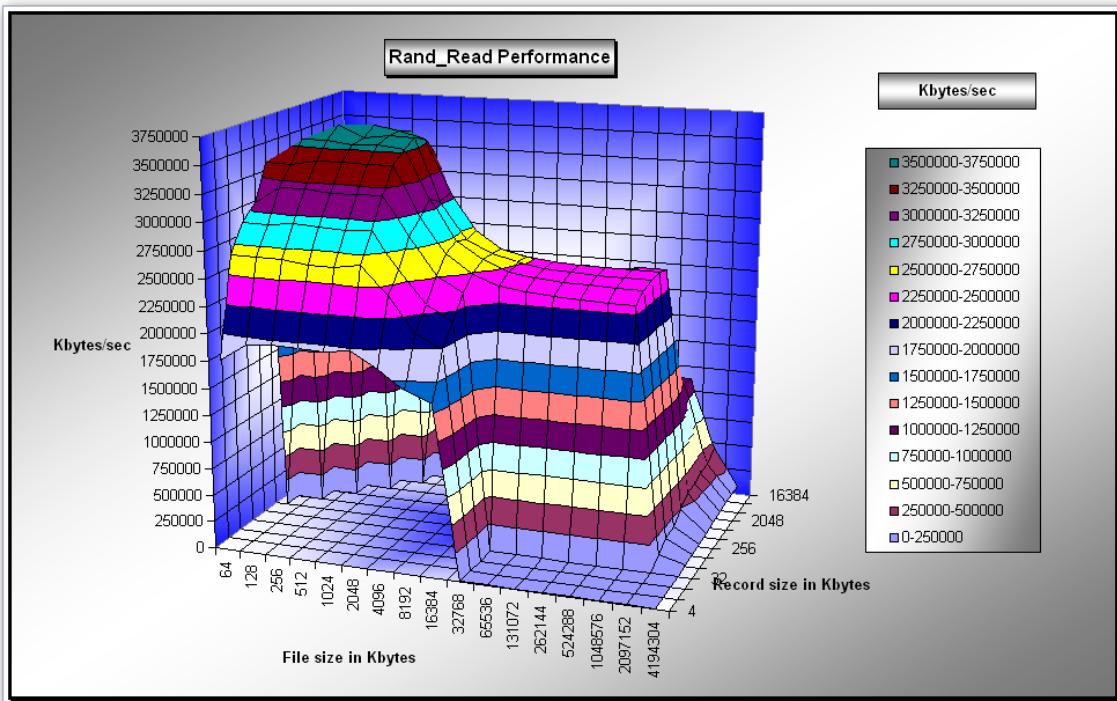


Figure 12

La *random-read* come in altri casi dà un profilo costante tra 2250000 e 2500000 Kbytes/sec.

REISERFS V3

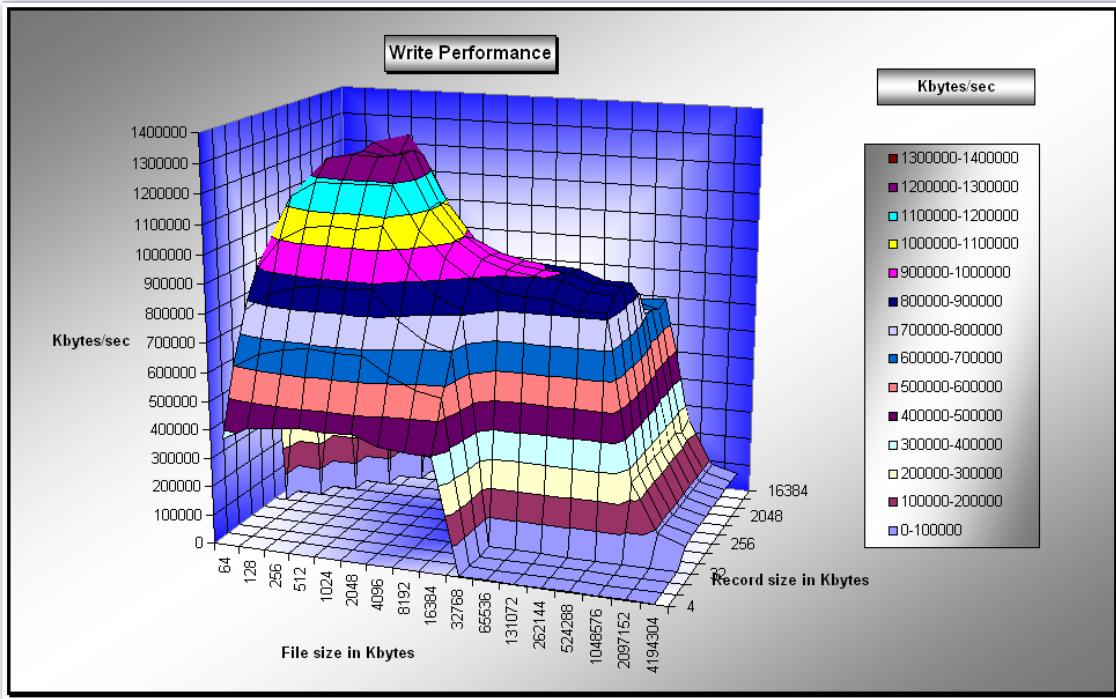


Figure 13

A parte il primo tratto, dà un andamento quasi costante con un throughput tra 800000 e 900000 Kbytes/sec.

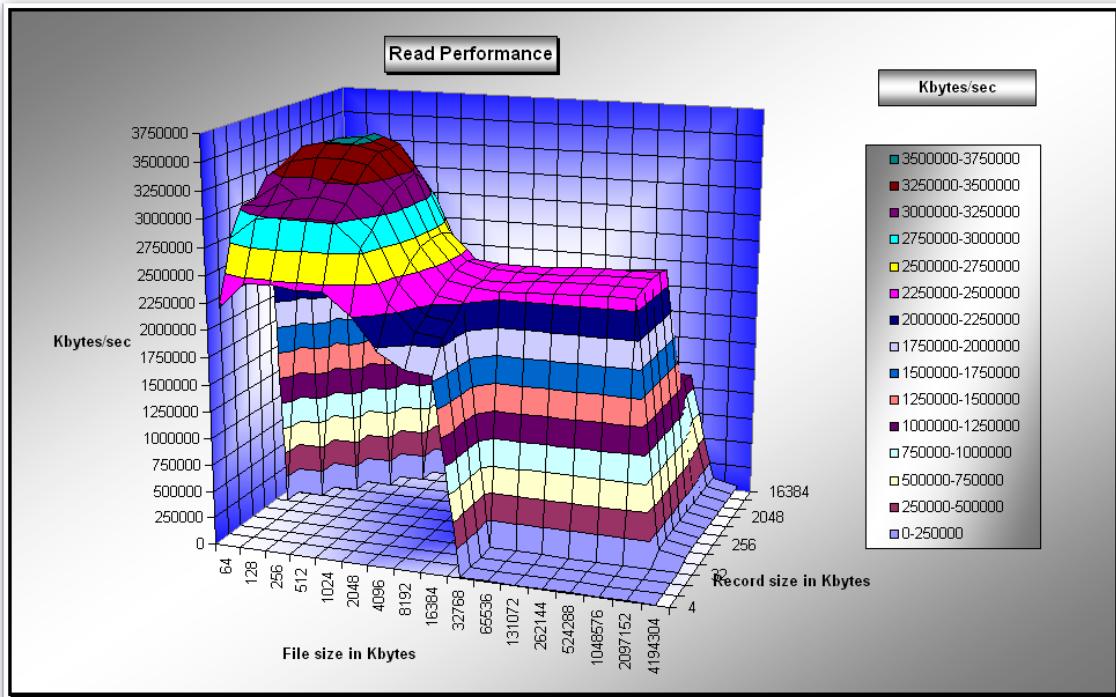


Figure 14

Anche qui si ha, a parte l'andamento a sella, un profilo quasi costante nel range tra 2250000 e 2500000 Kbytes/sec.

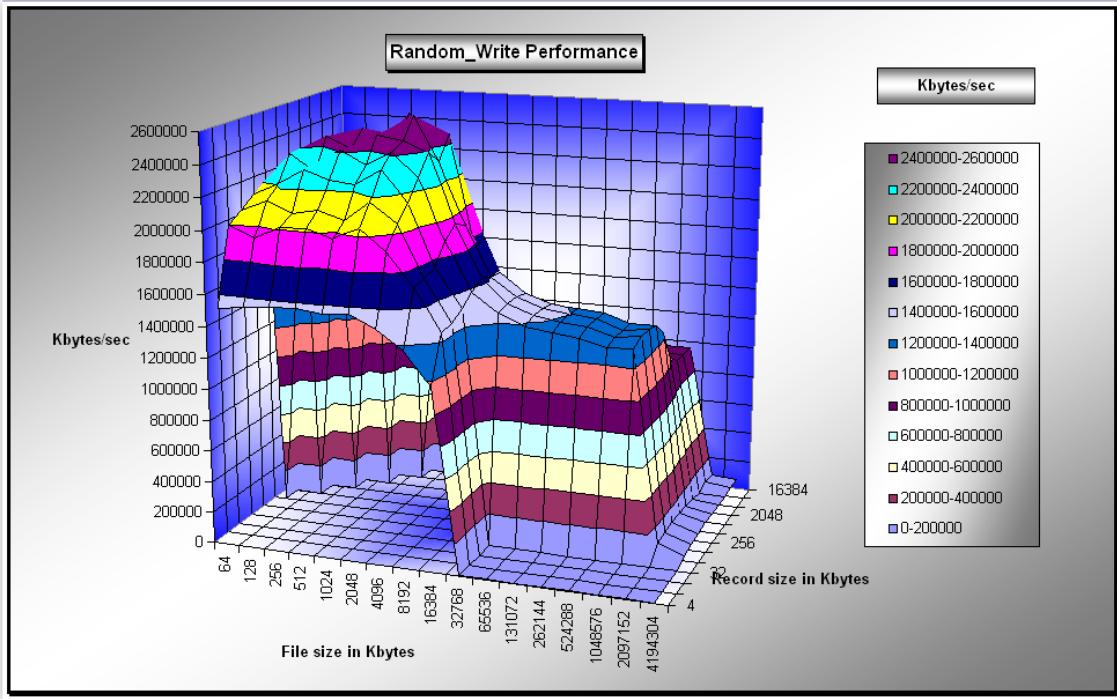


Figure 15

Qui si ha un andamento deprezzabile al di là di 8192 Kbytes.

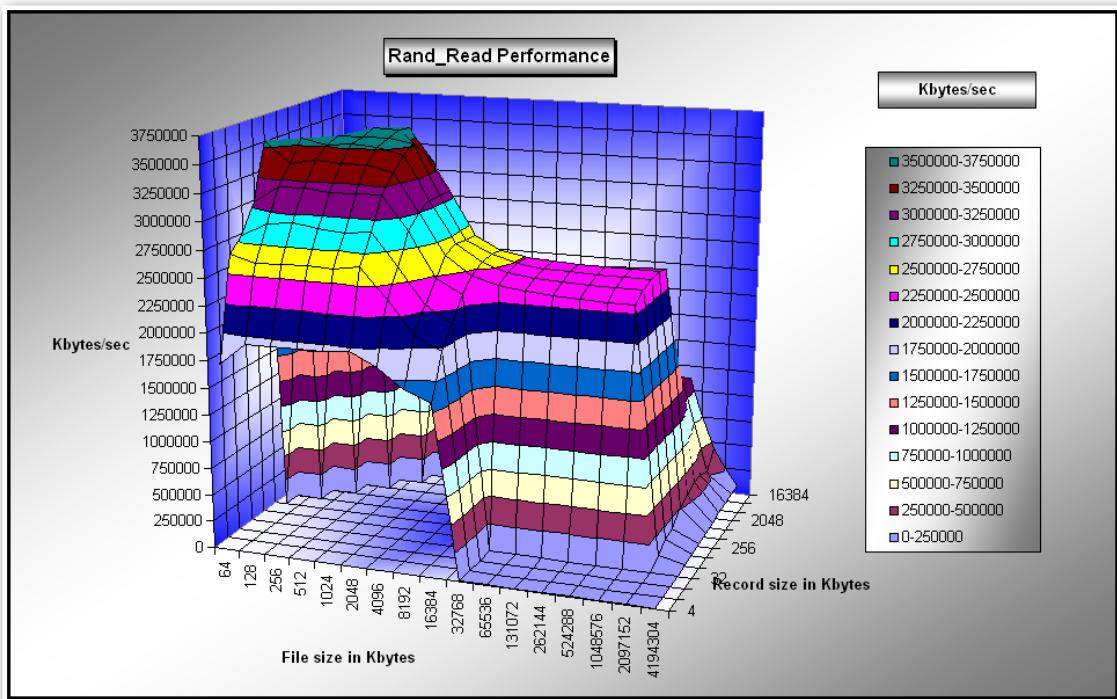


Figure 16

Invece nella *random-read* si ha un comportamento simile a quelli già visti.

XFS

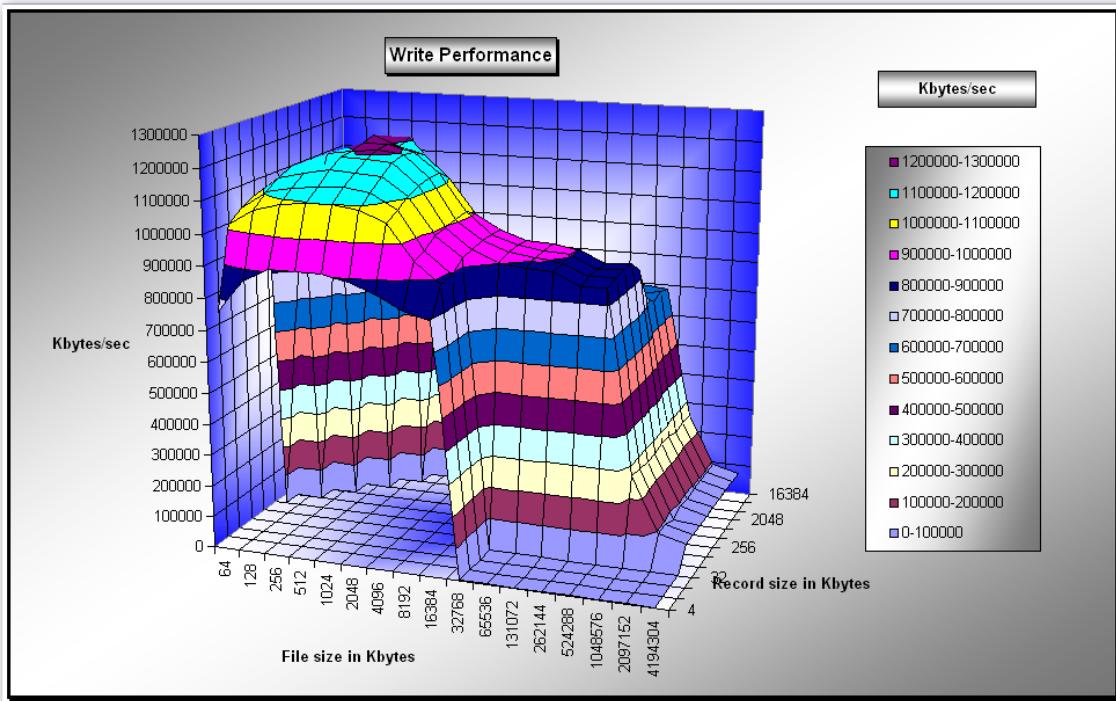


Figure 17

La XFS ha un migliore andamento nel range tra 128 e 8192 Kbytes.

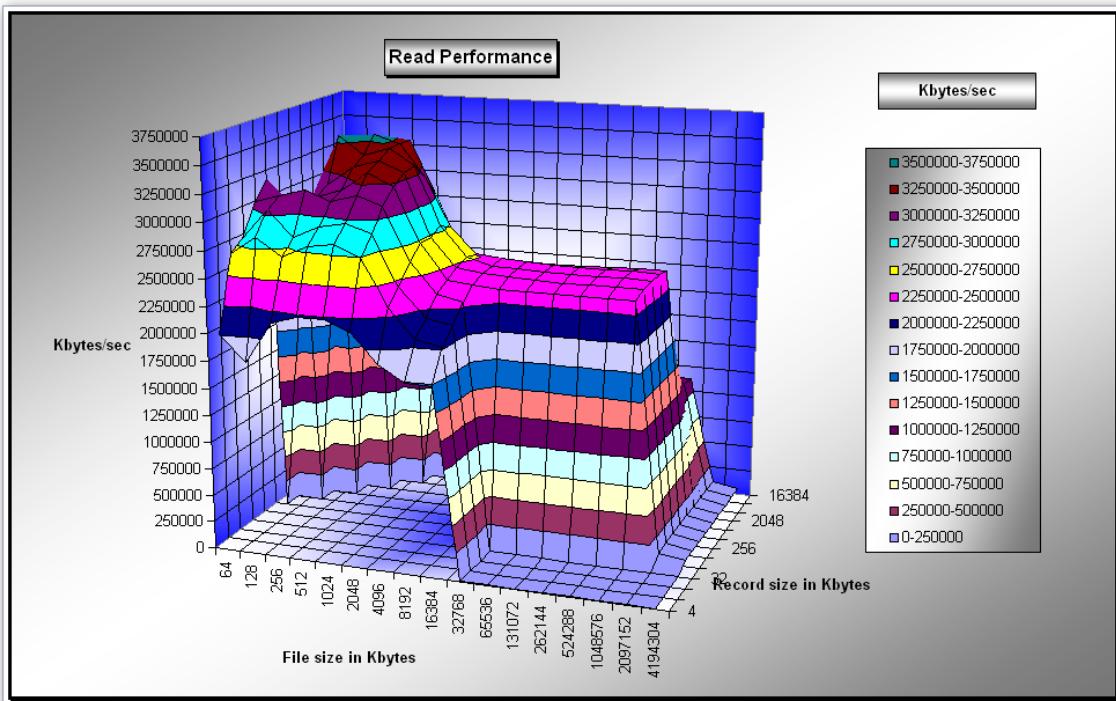


Figure 18

Nella *read*, a parte il tratto tra 64 e 1024, si ha un profilo costante pari a 2250000 e 2500000 Kbytes/sec.

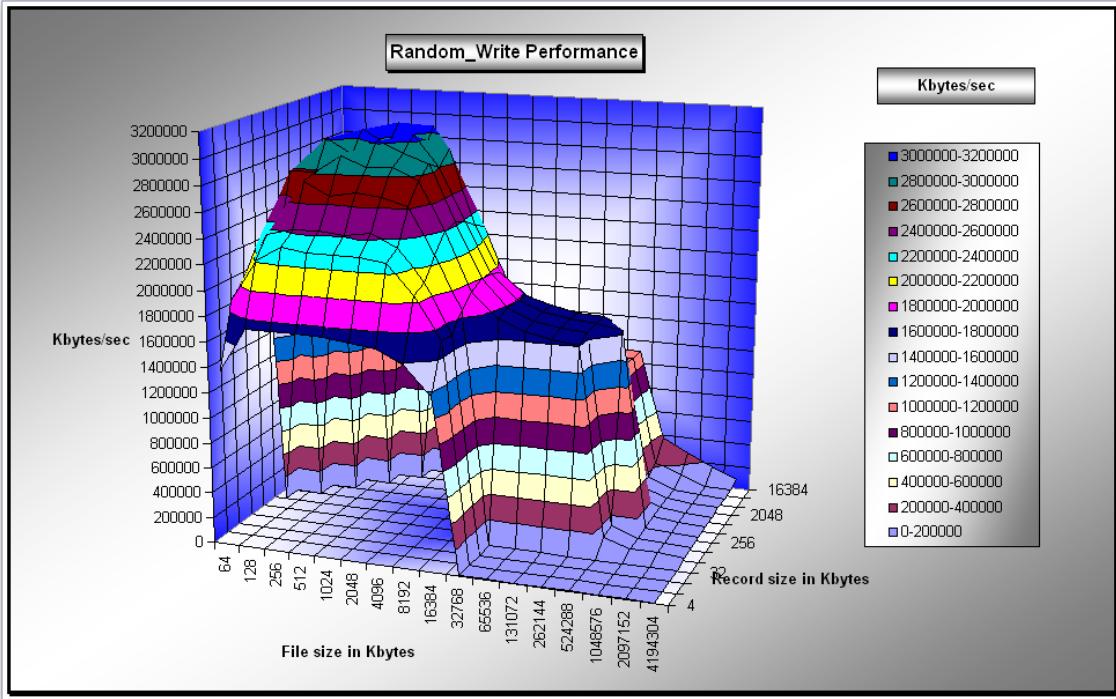


Figure 19

Nella *random-write* si evidenzia un buon profilo nel range tra 128 e 8192 Kbytes.

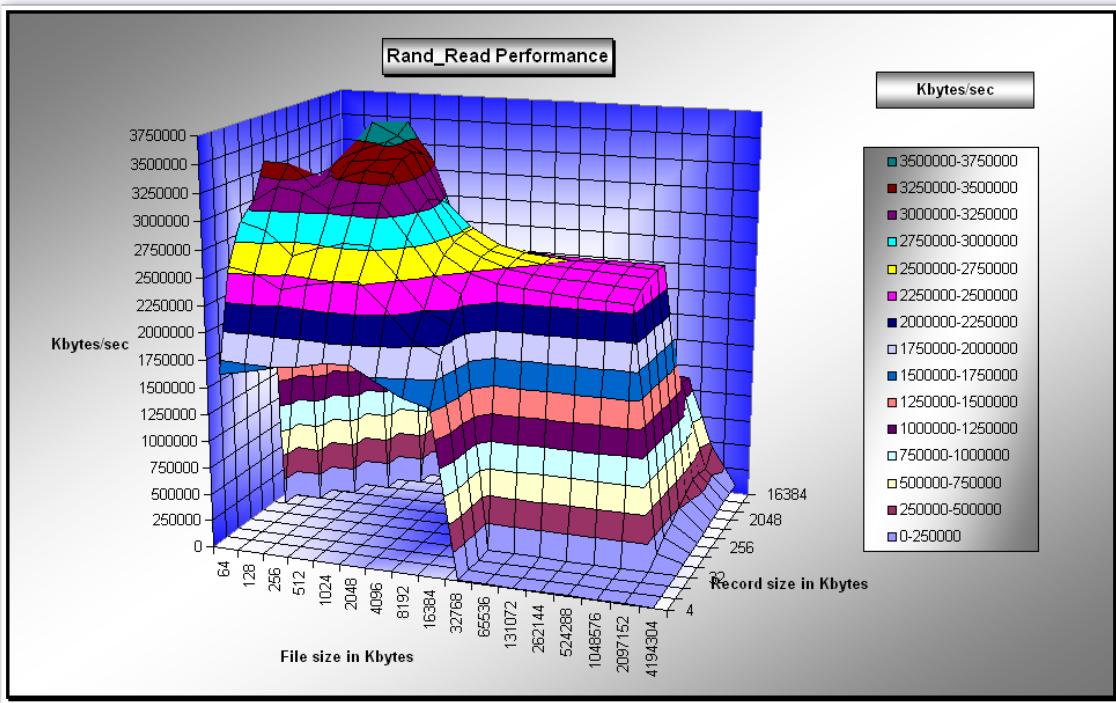


Figure 20

Anche la *random-read* permette d'avere un profilo tra 2250000 e 2500000 Kbytes/sec per qualsiasi file size.

RAMFS con dimensione 1G

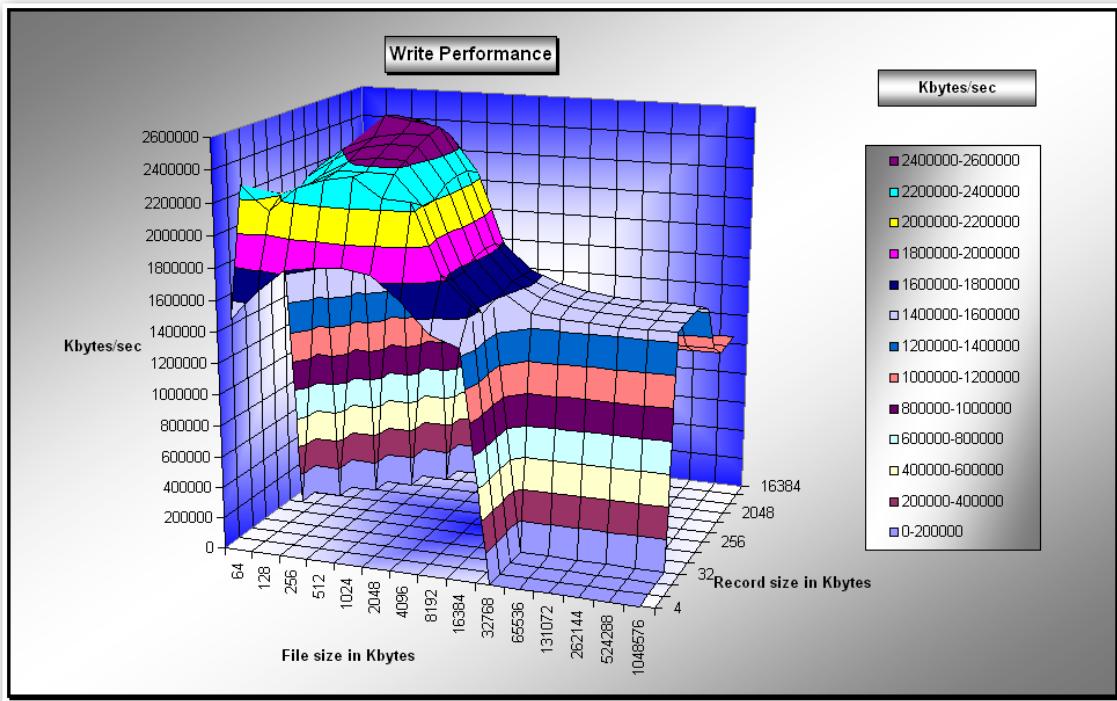


Figure 21

La RAMFS dà migliori performance nell'intervallo tra 128 e 4096 Kbytes.

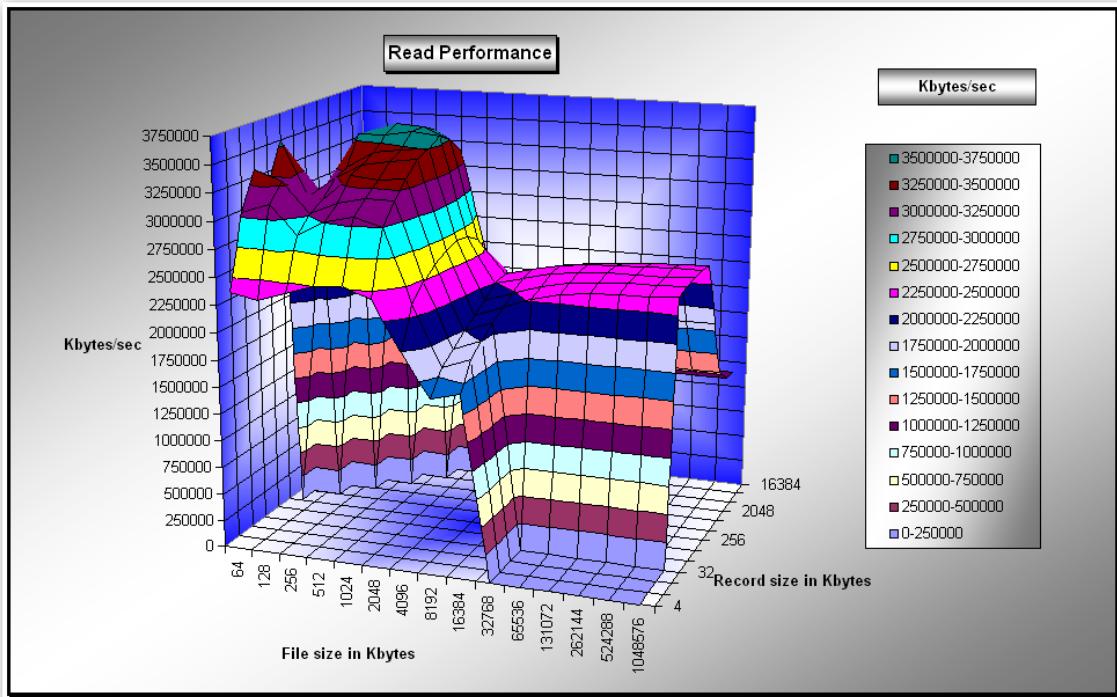


Figure 22

Le *read* è abbastanza singolare. Le migliori performance si ottengono tra 128 e 4096 Kbytes.

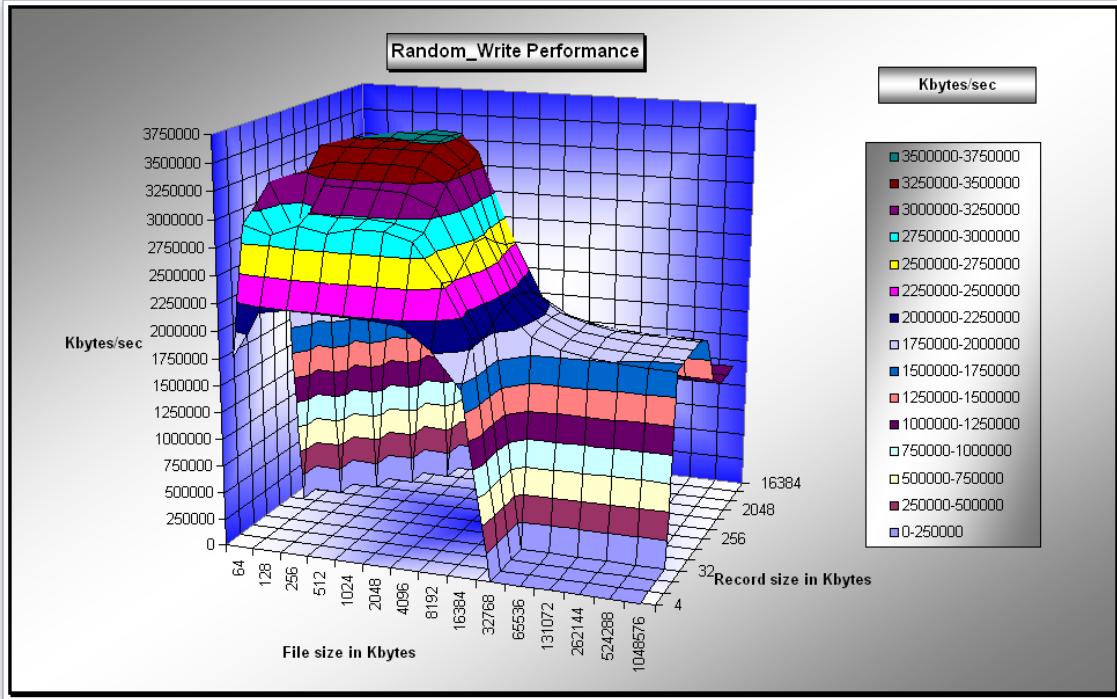


Figure 23

Anche nella *random-write* è solo in un intervallo tra 128 e 4096 che si ottengono migliori throughput.

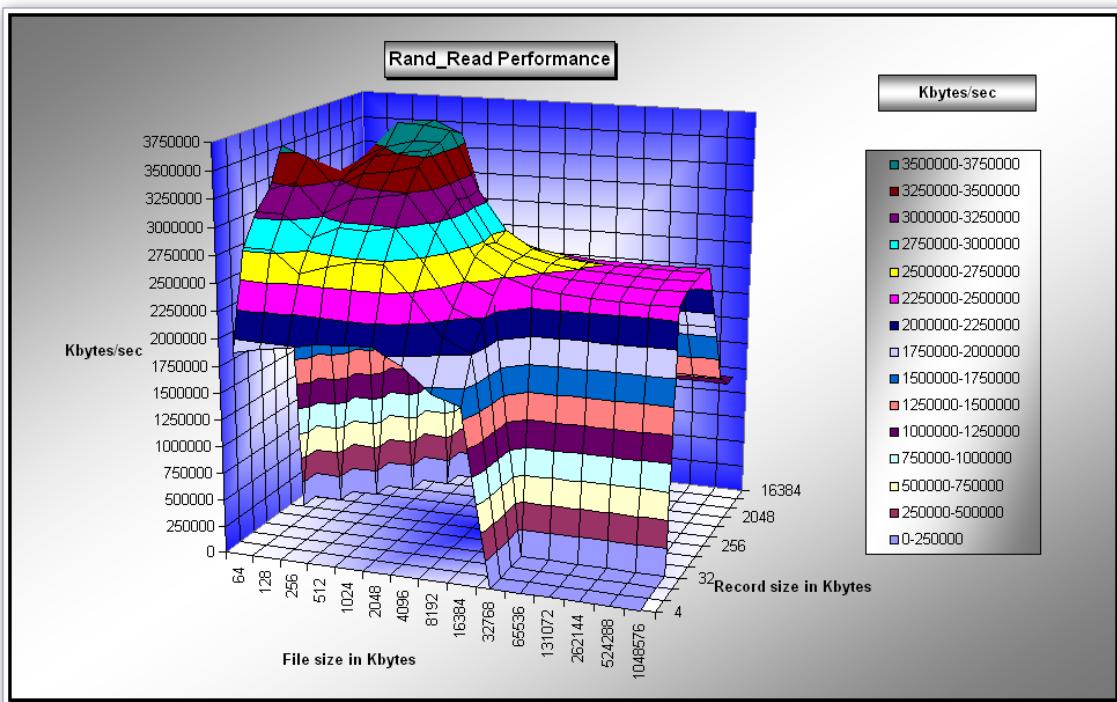


Figure 24

La random-read ha anch'esso un profilo costante tra 2250000 e 2500000 Kbytes/sec per ogni file size.

Risultati Bonnie++

Con il tool Bonnie++ è stata fatta una sola sequenza di test.
Questi test sono meno apprezzabili rispetto a quelli ottenuti con IOzone, ma si includono per completezza.

Con l'indice "n" si indica il numero di file come multiplo di 1024.

n=1

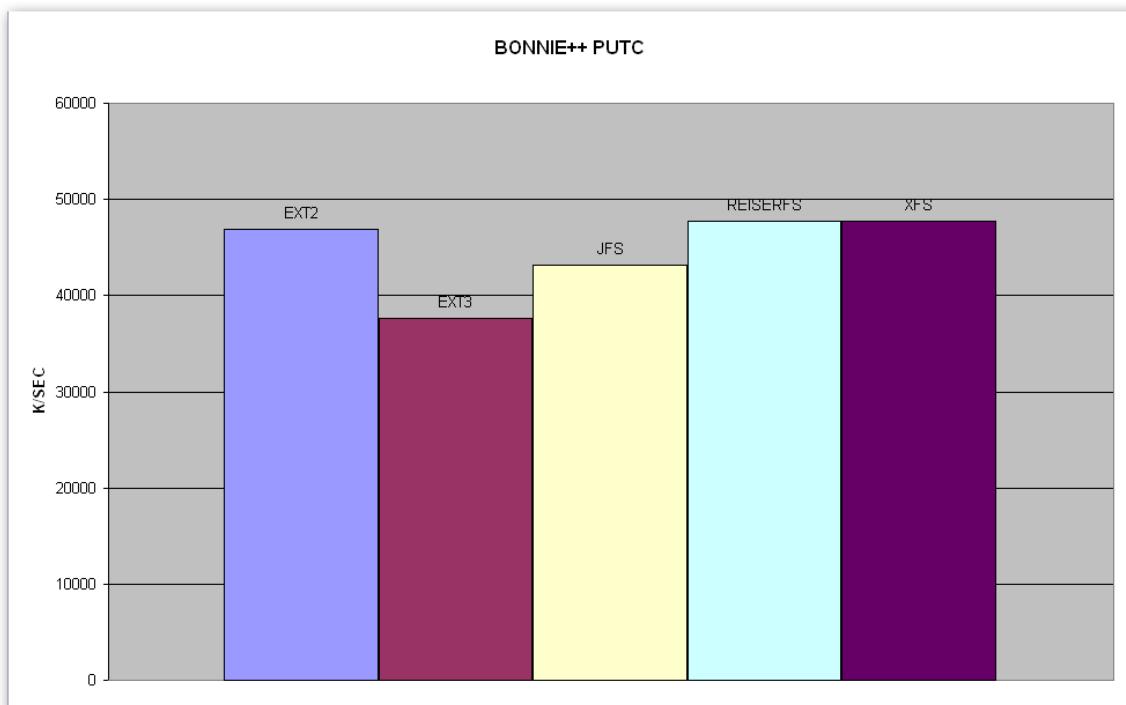


Figure 25

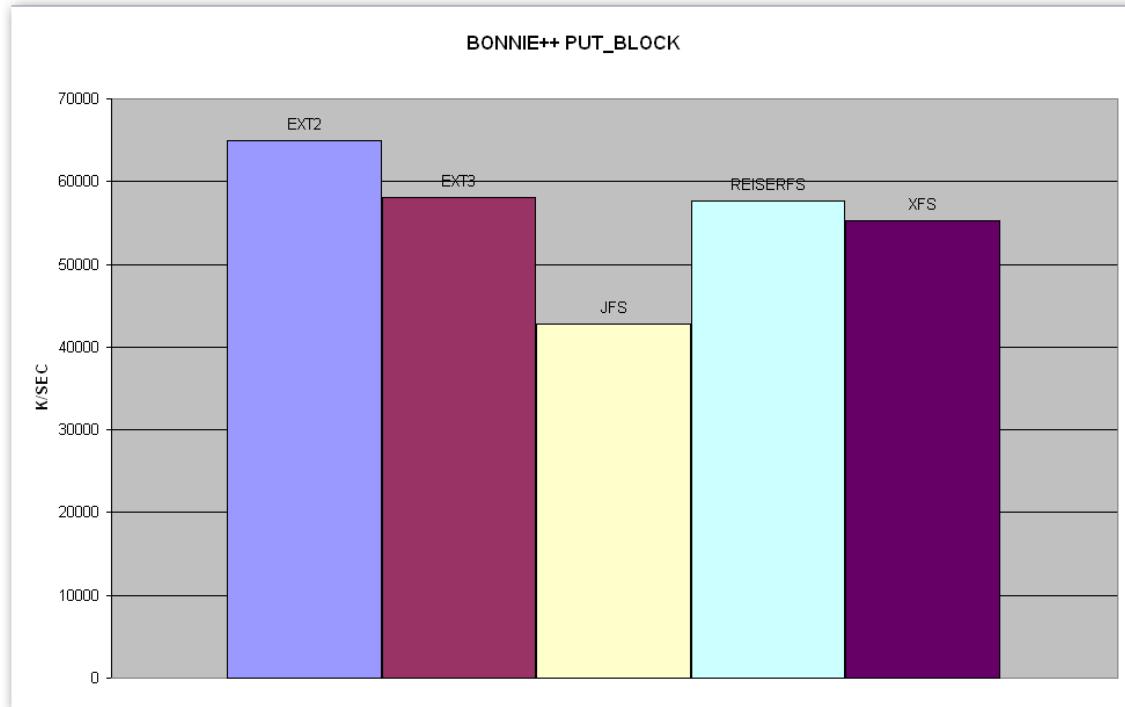


Figure 26

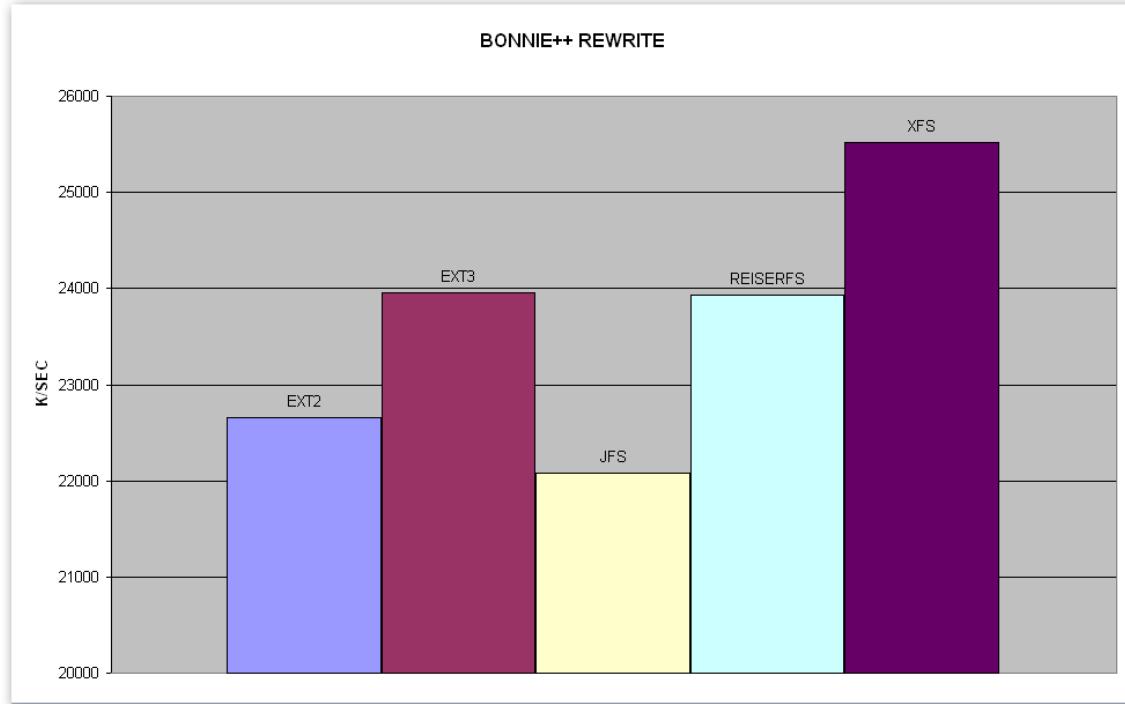


Figure 27

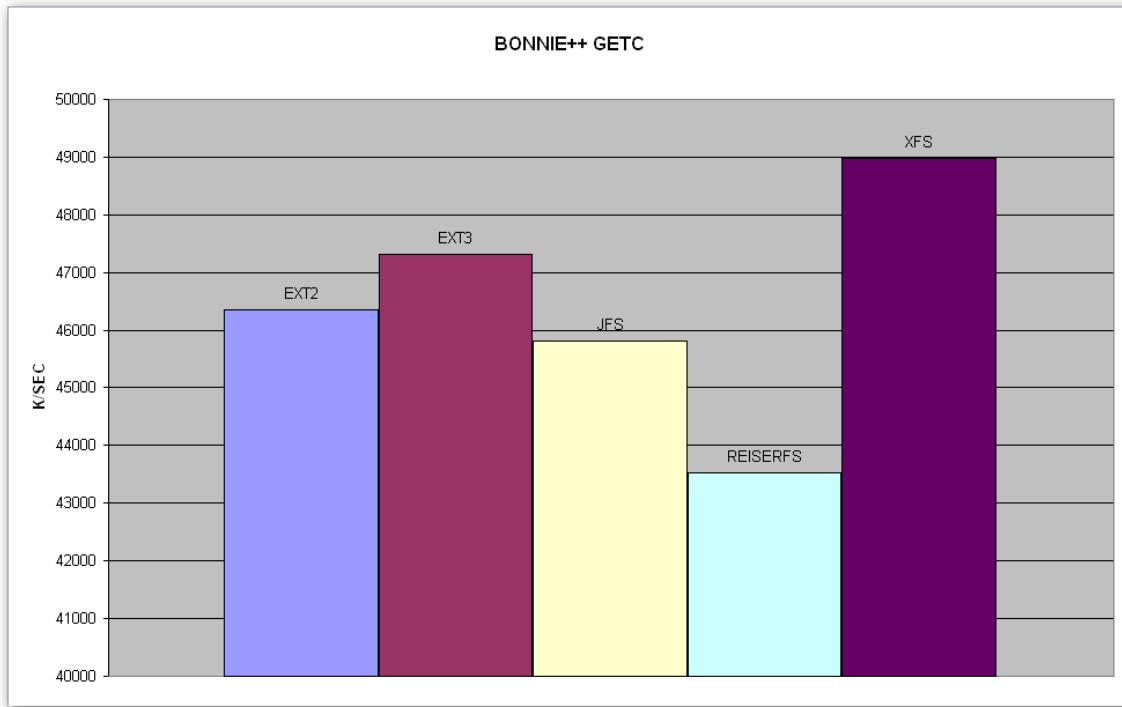


Figure 28

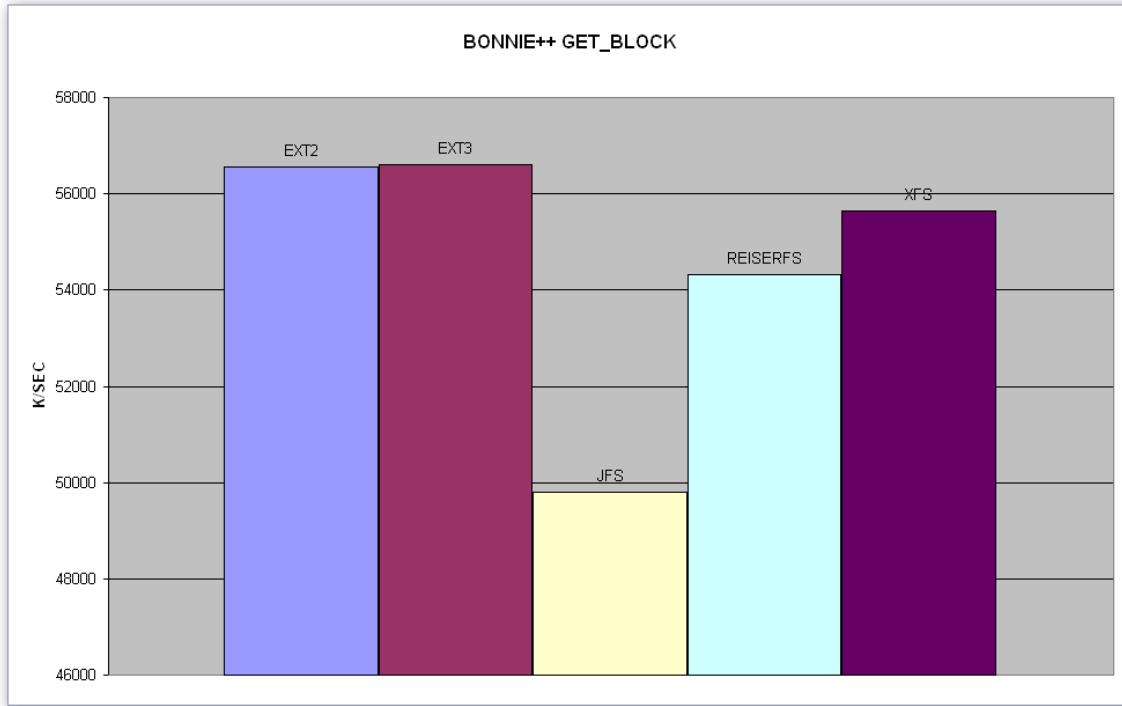


Figure 29

n=1024

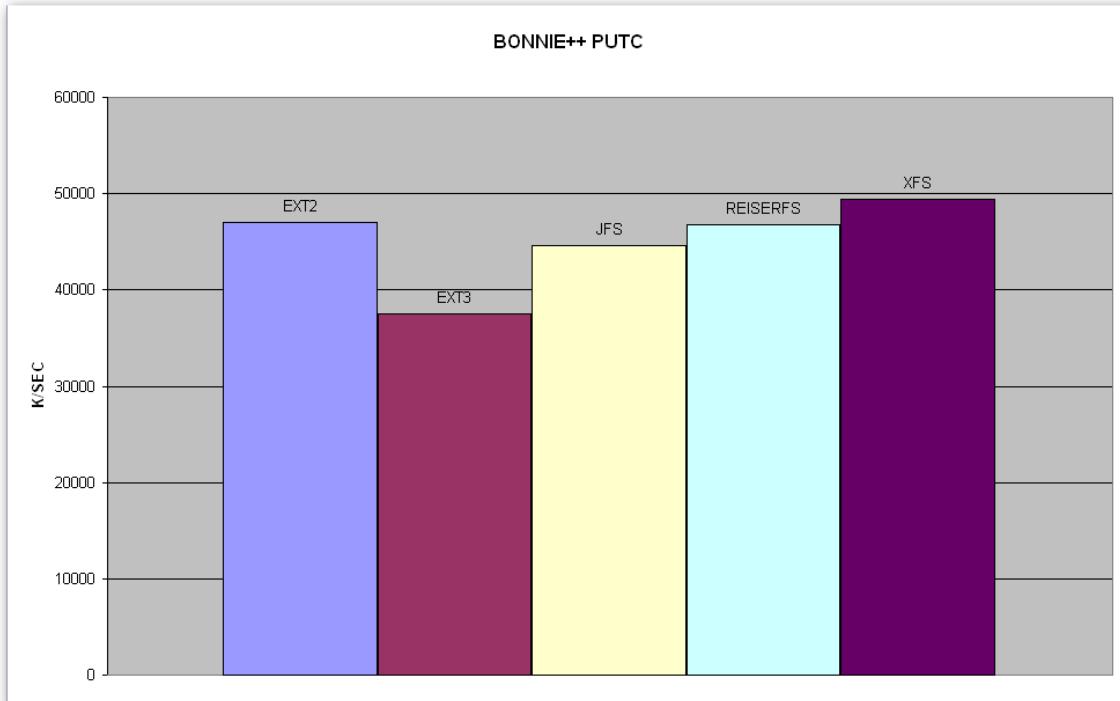


Figure 30

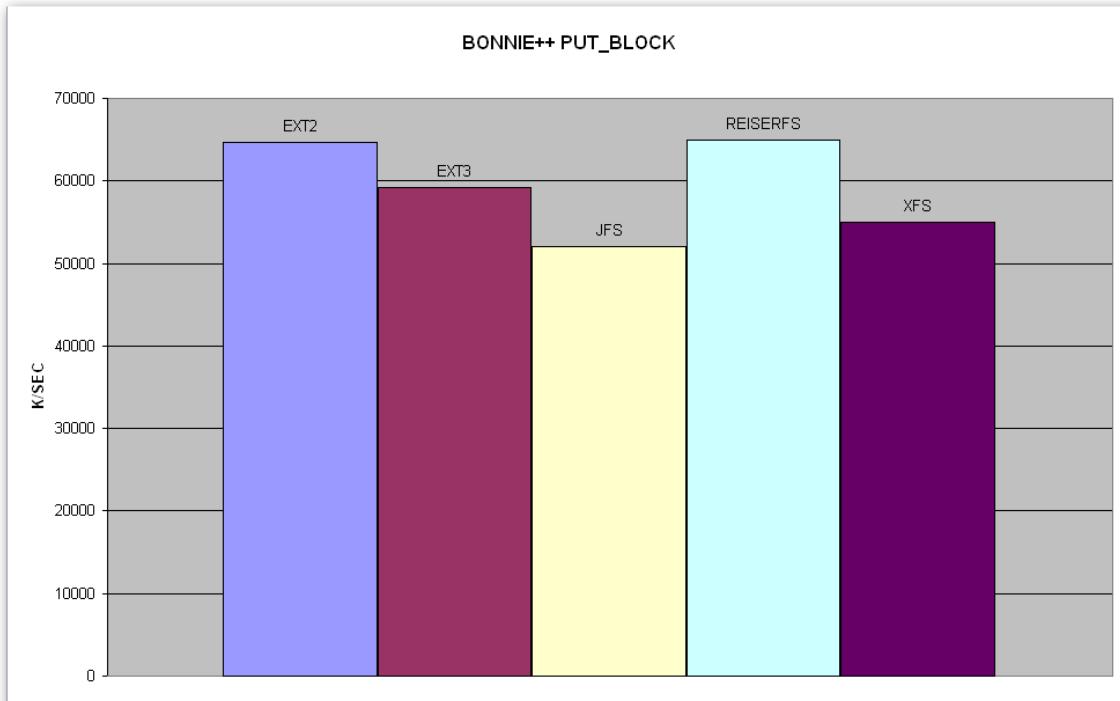


Figure 31

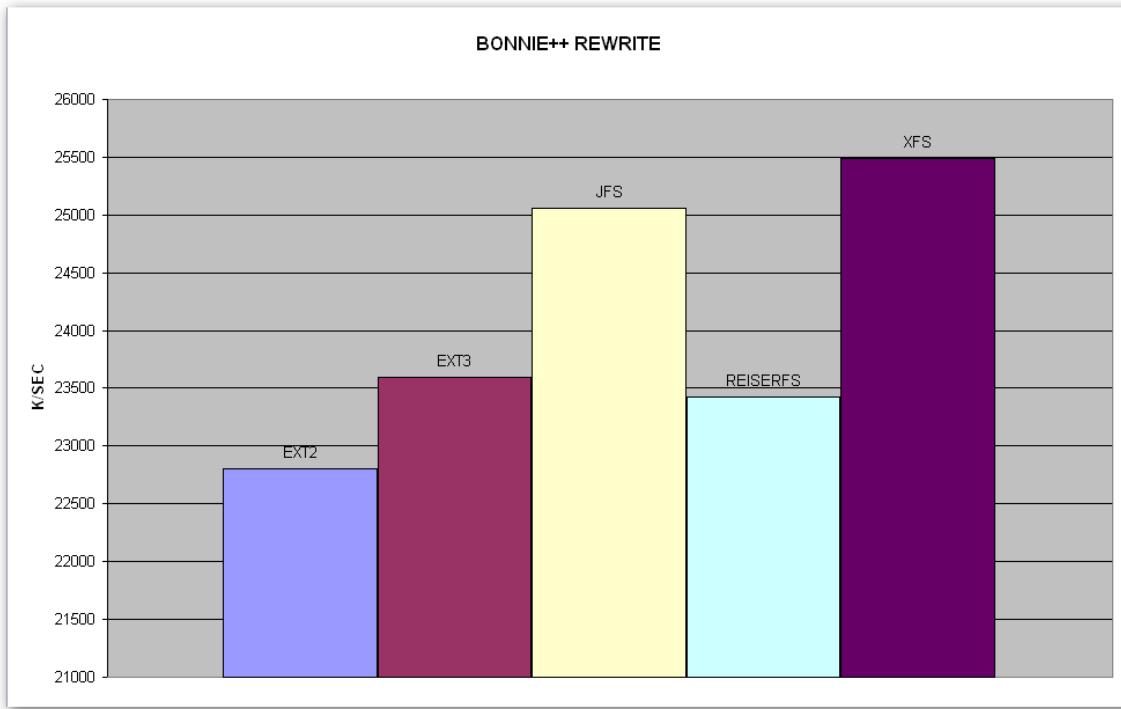


Figure 32

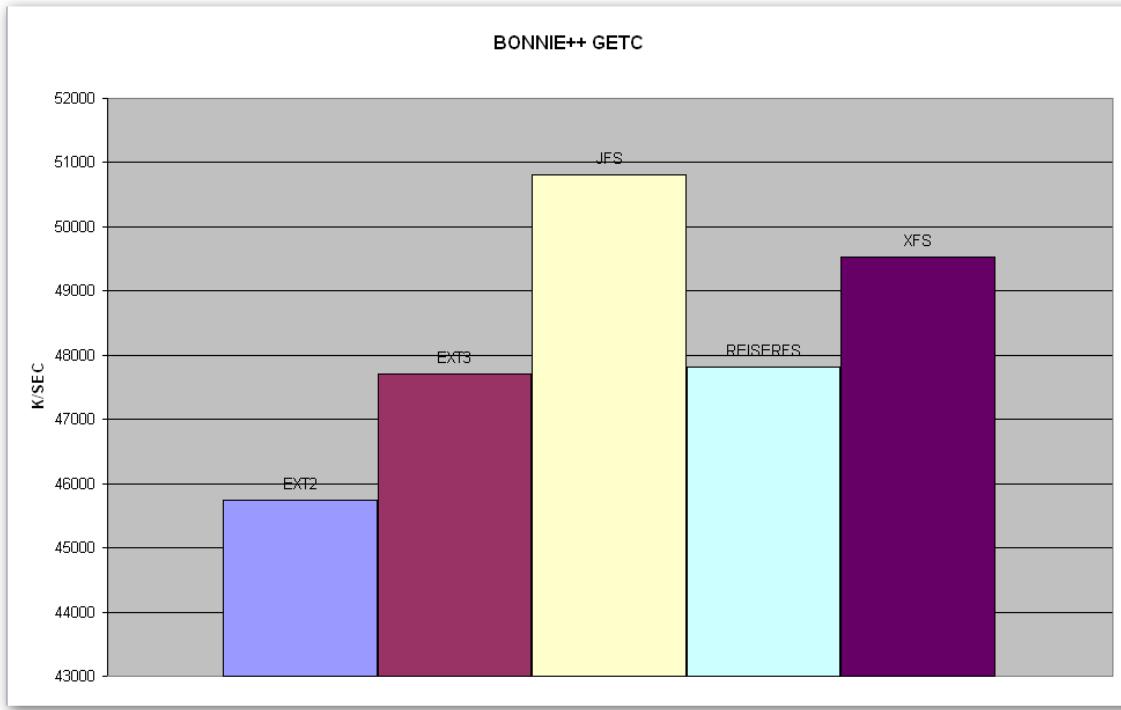


Figure 33

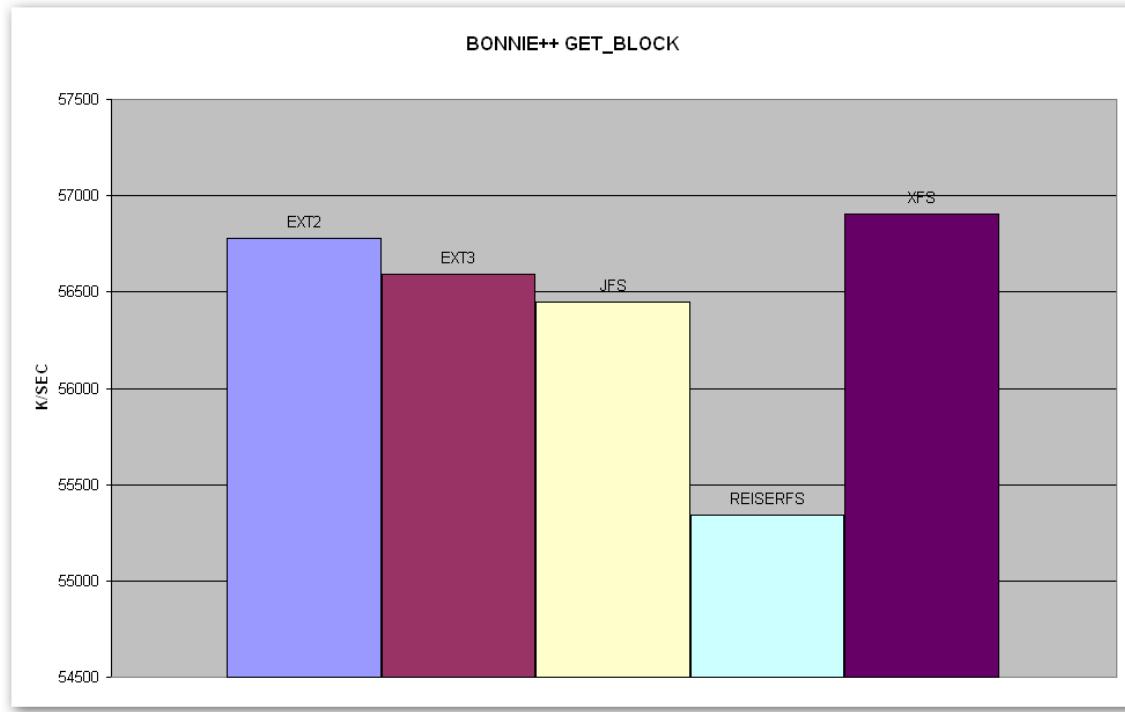


Figure 34

Conclusioni

I due tool utilizzati al fine di dare una caratterizzazione del filesystem da utilizzare danno apprezzamenti completamente differenti. Mentre IOzone dà una serie di dati per cui si rende necessario l'utilizzo di una macro excel per poter ottenere una sua rappresentazione grafica. Nel caso di Bonnie++ si ha un semplice valore numerico che dà un'informazione macroscopica.

Da queste semplici considerazioni si utilizzerà IOzone come elemento di valutazione.

L'elemento discriminante è la tipologia d'uso del filesystem. Nel nostro caso l'obiettivo è la memorizzazione di sessioni PCAP ottenute mediante sniffing diretto sulla rete wired.

Considerando che un throughput di 1Gbit/s è pari a 131072 Kbytes/sec risulta che tutti i profili potrebbero adattarsi allo scopo: non creare alcun collo di bottiglia alla capacità dello sniffer nel memorizzare i file pcap, a parte la EXT3 che in write penalizza fortemente.

In generale tutte le prove hanno dato risultati abbastanza apprezzabili. Le dimensioni dei file PCAP in genere non dovrebbero superare le dimensioni viste con i test.

In letteratura i risultati di altri benchmark hanno portato ad una scelta individuale più che ad una scelta dettata dalla evidenza dei risultati. Questo perché nel complesso si equivalgono. Per ora si utilizzerà un filesystem XFS per poi verificare in runtime le sue prestazioni nel momento in cui si utilizzerà lo sniffer vero e proprio.

Bibliografia

- [1] IOzone Filesystem Benchmark URL: <http://www.iozone.org/>
- [2] Bonnie++ benchmark suite URL: <http://www.coker.com.au/bonnie++/>
- [3] Benchmarking Filesystems Part II By Justin Piszcz URL:
<http://linuxgazette.net/122/piszcz.html>