MooseFS 权威指南

——分布式文件系统一站式解决方案

内容目录

1 前言

1.1 原文及 QQ 群

1. 原文作者：shinelian　原文地址：http://bbs.chinaunix.net/viewthread.php?tid=1644309

2. QQ 群：102082446　　分布式文件系统 专业群（不求人多，只求专业） 通关密码：i love cuer!

1.2 感谢

i. 特别感谢田逸的文档 http://sery.blog.51cto.com/10037/263515

ii. 特别感谢网友:灵犀 ，和官方开发人员沟通，并提供目前的一些官方文档

iii. 特别感谢网友:tt，灵犀，流云风，hzqbbc 在 qq 群内对广大爱好者分享宝贵经验。

iv. 特别感谢存储专家-《大话存储》的作者：冬瓜头 ，在我进行性能测试的时候，对我进行的指导。

v. 一个不知道名字的哥们（看到请联系我）

1.3 架构图

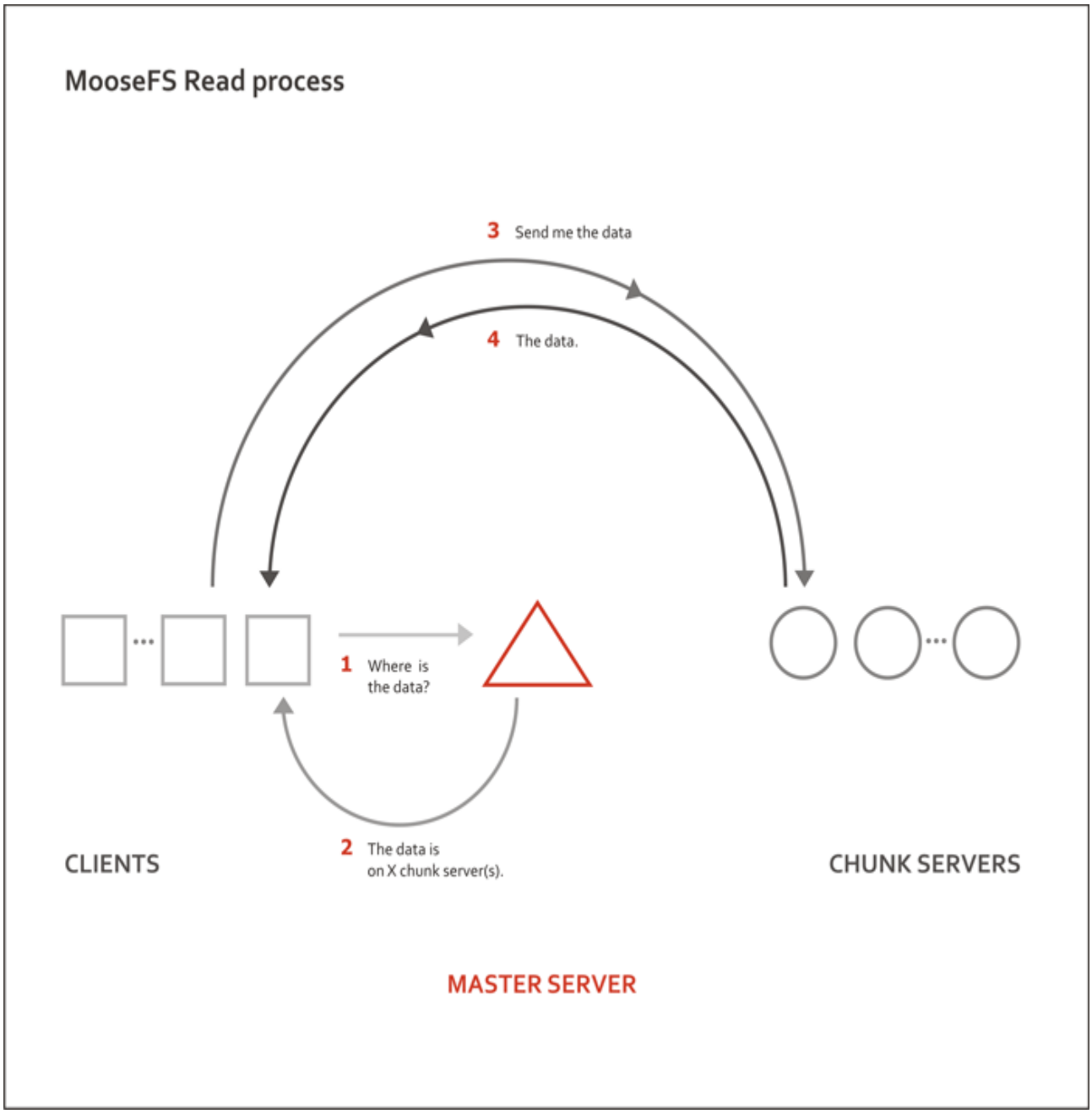


插图 1: MooseFS Read process

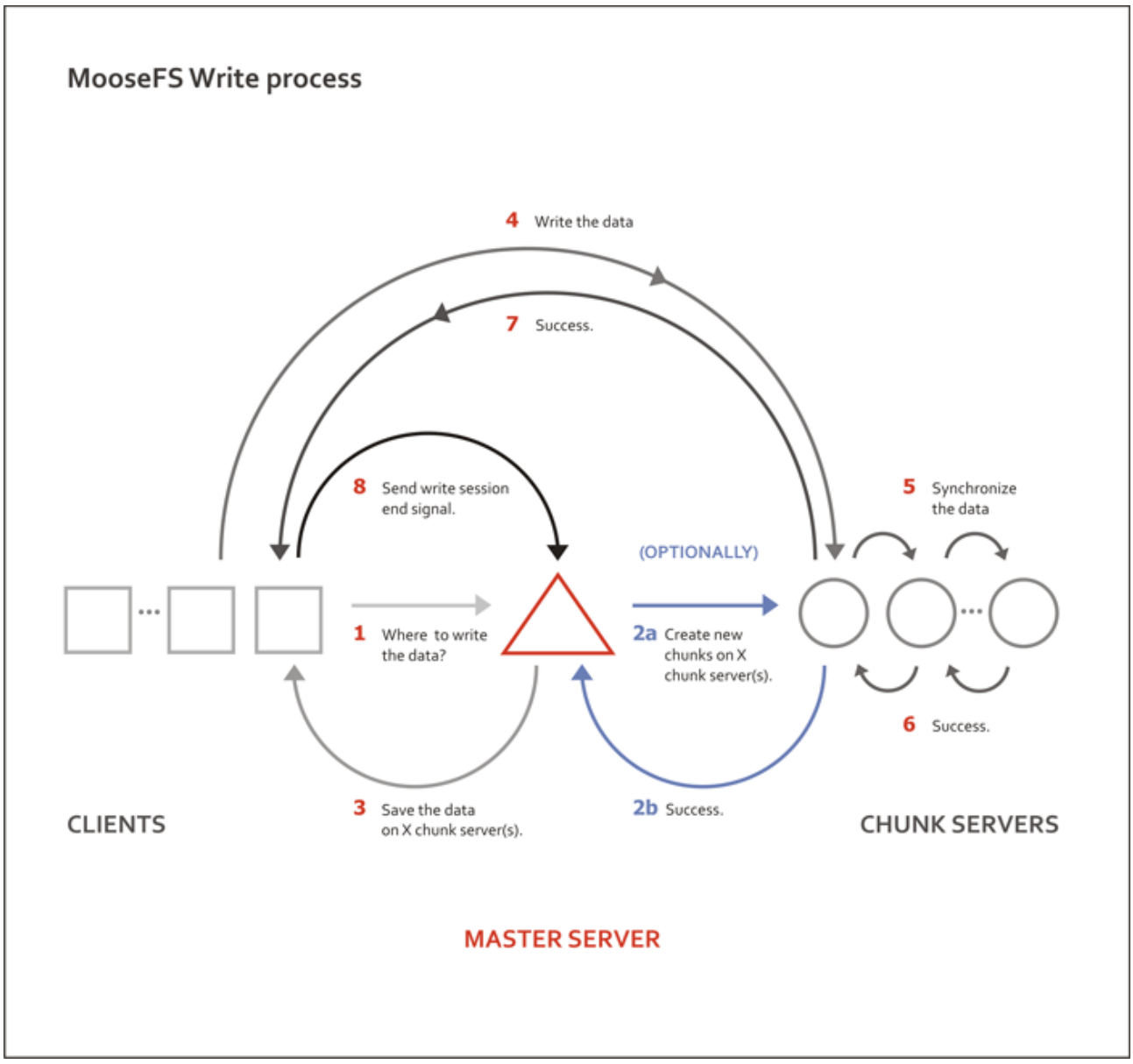


插图 2: MooseFS Write process

1.4 MooseFS 优点

i. Free(GPL)

ii. 通用文件系统，不需要修改上层应用就可以使用（那些需要专门 api 的 dfs 好麻烦哦！）。

iii. 可以在线扩容，体系架构可伸缩性极强。（官方的 case 可以扩到 70 台了！）

iv. 部署简单。（sa 们特别高兴，领导们特别 happy！）

v. 体系架构高可用，所有组件无单点故障。（您还等什么？）

vi. 文件对象高可用，可设置任意的文件冗余程度（提供比 raid1+0 更高的冗余级别），而绝对不会影响读或者写的性能，只会加速哦！）

vii. 提供 Windows 回收站的功能.（不怕误操作了，提供类似 oralce 的闪回等高级 dbms 的即时回滚特性，oralce 这些特性可是收费的哦！）

viii. 提供类似 Java 语言的 GC（垃圾回收）。

ix. 提供 netapp，emc，ibm 等商业存储的 snapshot 特性。

x. google filesystem 的一个 c 实现。（google 在前面开路哦！）

xi. 提供 web gui 监控接口。

xii. 提高随机读或写的效率（有待进一步证明）。

xiii. 提高海量小文件的读写效率（有待进一步证明）。

1.5 MooseFS 1.6 版本改进[1]

1. 修复 1.5.x 中在大批量操作时打开文件过多的 bug。这个错误也在我们此次测试的时候遇到，报的错误说是打开的文件过多，造成 chunker server 的链接错误。虽然后来的测试中一直想模拟出来这个问题，但是一直无法模拟出来。在 1.6.x 中解决此问题，就解决了很大的问题。
2. 新增加了 masterlogger 服务器。这是在 1.5.x 中所没有的，就是做了 master 服务器的冗余，进一步的加强的 master 服务器的稳定性。在 mfs 体系中 master 是要求最稳定以及性能要求最高的，因此务必保证 master 的稳定。
3. 修改 1.5.x 中存在的对于坏块的修复功能。在 mfs1.5.x 中遇到 chunker 坏块校验，错误比较多的是很往往导致 master 将出现坏块的 chunker 自动的剔除出去的情况，此次增加了对坏块的修复功能，很方便的进行修复，简化对坏块的处理功能。
4. 对 metadata 和 changelog 的新认识。之前认为 changelog 记录的是文件的操作，定期的像数据库的日志一样归档到 metadata 中。发现上面的理解存在误区，真正的是 changelog 中记录了对文件的操作，metadata 记录文件的大小和位置。因此 metadata 是比较重要的，在进行修复的过程中是采用 metadata 和最后一次的 changelog 进行修复的。
5. MFS 文档中明确指出对于内存和磁盘大小的要求。【In our environment (ca. 500 TiB, 25 million files, 2 million folders distributed on 26 million chunks on 70 machines) the usage of chunkserver CPU (by constant file transfer) is about 15-20% and chunkserver RAM usually consumes about 100MiB (independent of amount of data).

   The master server consumes about 30% of CPU (ca. 1500 operations per second) and 8GiB RAM. CPU load depends on amount of operations and RAM on number of files and folders.】
6. 指出了在测试的过程中多个 chunker 并不影响写的速度，但是能加快读的速度。在原来的基础上增加一个 chunker 时，数据会自动同步到新增的 chunker 上以达到数据的平衡和均衡。

---

1 mfs 1.6.x 的 User Guides 和 FAQ，并和灵犀沟通对文档中不理解的地方，就理解不一致的地方达成一致。特别感谢 qq 群内网友 流云风 和 灵犀

1.6 Web GUI 监控界面

MFS Moose File System

Info − | Servers + | Disks + | Exports + | Mounts + | Operations + | Master Charts + | Server Charts +

W3C CSS | W3C XHTML 1.0

Info

version	total space	avail space	trash space	trash files	reserved space	reserved files	all fs objects	directories	files	chunks	all chunk copies	regular chunk copies
1.6.10	513 TiB	52 TiB	15 TiB	692300	366 GiB	2050	27365692	2138276	25219160	26024972	52288381	52143488

All chunks state matrix (counts 'regular' hdd space and 'marked for removal' hdd space : switch to 'regular')

goal	valid copies											
	0	1	2	3	4	5	6	7	8	9	10+	all
0	3506	45	1381	224	-	-	-	-	-	-	-	5156
1	-	-	-	-	-	-	-	-	-	-	-	0
2	-	900904	23922954	132598	-	-	-	-	-	-	-	24956456
3	-	2	57005	999128	7225	-	-	-	-	-	-	1063360
4	-	-	-	-	-	-	-	-	-	-	-	0
5	-	-	-	-	-	-	-	-	-	-	-	0
6	-	-	-	-	-	-	-	-	-	-	-	0
7	-	-	-	-	-	-	-	-	-	-	-	0
8	-	-	-	-	-	-	-	-	-	-	-	0
9	-	-	-	-	-	-	-	-	-	-	-	0
10+	-	-	-	-	-	-	-	-	-	-	-	0
all 1+	0	900906	23979959	1131726	7225	0	0	0	0	0	0	26019816

Chunk operations info

loop time		deletions				replications	
start	end	invalid	unused	disk clean	over goal	under goal	rebalance
Fri Dec 18 11:52:15 2009	Fri Dec 18 11:57:15 2009	0/0	7424/7441	0/0	0/0	0/962975	0

Filesystem check info

check loop start time	check loop end time	files	under-goal files	missing files	chunks	under-goal chunks	missing chunks
Fri Dec 18 10:55:27 2009	Fri Dec 18 11:55:30 2009	25210578	213730	0	26893102	240784	0

MFS Moose File System

Info + | Servers − | Disks + | Exports + | Mounts + | Operations + | Master Charts + | Server Charts +

W3C CSS | W3C XHTML 1.0

Chunk Servers

#	host	ip	port	version	'regular' hdd space				'marked for removal' hdd space			
					chunks	used	total	% used	chunks	used	total	% used
1	szafa2.local.gem.pl	192.168.1.102	9422	1.6.10	645443	5.9 TiB	6.3 TiB	93.52	0	0 B	0 B	-
2	szafa3.local.gem.pl	192.168.1.103	9422	1.6.10	962727	8.7 TiB	9.4 TiB	92.04	0	0 B	0 B	-
3	szafa5.local.gem.pl	192.168.1.105	9422	1.6.10	854525	7.4 TiB	8.1 TiB	92.04	0	0 B	0 B	-
4	szafa6.local.gem.pl	192.168.1.106	9422	1.6.10	713573	6.3 TiB	6.7 TiB	93.40	0	0 B	0 B	-
5	szafa7.local.gem.pl	192.168.1.107	9422	1.6.10	706907	6.2 TiB	6.7 TiB	93.03	0	0 B	0 B	-
6	szafa10.local.gem.pl	192.168.1.110	9422	1.6.10	428260	3.9 TiB	4.0 TiB	96.90	0	0 B	0 B	-
7	szafa14.local.gem.pl	192.168.1.114	9422	1.6.10	567994	5.1 TiB	5.4 TiB	95.56	0	0 B	0 B	-
8	szafa15.local.gem.pl	192.168.1.115	9422	1.6.10	609013	5.5 TiB	5.8 TiB	94.62	0	0 B	0 B	-
9	szafa17.local.gem.pl	192.168.1.117	9422	1.6.10	1324375	11 TiB	12 TiB	91.70	0	0 B	0 B	-
10	szafa18.local.gem.pl	192.168.1.118	9422	1.6.10	964767	8.6 TiB	9.4 TiB	91.56	0	0 B	0 B	-
11	szafa26.local.gem.pl	192.168.1.126	9422	1.6.10	609936	5.4 TiB	5.8 TiB	93.22	0	0 B	0 B	-
12	szafa27.local.gem.pl	192.168.1.127	9422	1.6.10	379435	3.5 TiB	3.6 TiB	96.55	0	0 B	0 B	-
13	szafa29.local.gem.pl	192.168.1.129	9422	1.6.10	646154	5.8 TiB	6.3 TiB	93.03	0	0 B	0 B	-
14	szafa30.local.gem.pl	192.168.1.130	9422	1.6.10	692022	6.2 TiB	6.7 TiB	92.24	0	0 B	0 B	-
15	szafa31.local.gem.pl	192.168.1.131	9422	1.6.10	618408	5.4 TiB	5.8 TiB	93.22	0	0 B	0 B	-
16	szafa33.local.gem.pl	192.168.1.133	9422	1.6.10	381712	3.5 TiB	3.6 TiB	97.02	0	0 B	0 B	-
17	szafa34.local.gem.pl	192.168.1.134	9422	1.6.10	366129	3.3 TiB	3.4 TiB	97.78	0	0 B	0 B	-
18	szafa35.local.gem.pl	192.168.1.135	9422	1.6.10	475457	4.3 TiB	4.5 TiB	95.80	0	0 B	0 B	-
19	szafa37.local.gem.pl	192.168.1.137	9422	1.6.10	609786	5.5 TiB	5.8 TiB	94.11	0	0 B	0 B	-
20	szafa38.local.gem.pl	192.168.1.138	9422	1.6.10	602560	5.4 TiB	5.8 TiB	93.28	0	0 B	0 B	-
21	szafa39.local.gem.pl	192.168.1.139	9422	1.6.10	832828	7.4 TiB	8.1 TiB	92.45	0	0 B	0 B	-
22	szafa42.local.gem.pl	192.168.1.142	9422	1.6.10	429128	3.9 TiB	4.0 TiB	95.84	0	0 B	0 B	-
23	szafa43.local.gem.pl	192.168.1.143	9422	1.6.10	429454	3.9 TiB	4.0 TiB	97.05	0	0 B	0 B	-
24	szafa45.local.gem.pl	192.168.1.145	9422	1.6.10	594986	5.0 TiB	5.4 TiB	93.65	0	0 B	0 B	-
25	szafa47.local.gem.pl	192.168.1.147	9422	1.6.10	882196	7.4 TiB	8.1 TiB	91.94	0	0 B	0 B	-
26	szafa49.local.gem.pl	192.168.1.149	9422	1.6.10	866405	7.4 TiB	8.1 TiB	91.68	144892	1.2 TiB	1.3 TiB	90.63
27	szafa50.local.gem.pl	192.168.1.150	9422	1.6.10	879130	7.5 TiB	8.1 TiB	92.73	0	0 B	0 B	-
28	szafa51.local.gem.pl	192.168.1.151	9422	1.6.10	804664	7.4 TiB	8.1 TiB	91.80	0	0 B	0 B	-

MFS Moose File System

Info + | Servers + | Disks − | Exports + | Mounts + | Operations + | Master Charts + | Server Charts +

W3C css | W3C XHTML 1.0

Disks

#	info				I/O stats last min (switch to hour,day)								space		
					transfer		max time (switch to avg)			# of ops					
	path	chunks	last error	status	read	write	read	write	fsync	read	write	fsync	used	total	used (%)
1	192.168.1.102:9422:/mnt/hd1/	91852	no errors	ok	12 MiB/s	17 MiB/s	421710 us	34182 us	995866 us	1167	4025	870	858 GiB	917 GiB	93.57
2	192.168.1.102:9422:/mnt/hd2/	92343	no errors	ok	31 MiB/s	36 MiB/s	68260 us	24654 us	231106 us	1426	761	36	858 GiB	917 GiB	93.55
3	192.168.1.102:9422:/mnt/hd3/	93422	no errors	ok	19 MiB/s	51 MiB/s	377363 us	45316 us	434964 us	982	541	8	857 GiB	917 GiB	93.51
4	192.168.1.102:9422:/mnt/hd4/	92892	no errors	ok	22 MiB/s	5.9 MiB/s	71920 us	11277 us	12772 us	987	7	4	857 GiB	917 GiB	93.49
5	192.168.1.102:9422:/mnt/hd5/	91158	no errors	ok	29 MiB/s	35 MiB/s	108001 us	19255 us	456473 us	1351	1615	35	857 GiB	917 GiB	93.51
6	192.168.1.102:9422:/mnt/hd6/	90544	no errors	ok	15 MiB/s	20 MiB/s	249753 us	20692 us	362826 us	1685	1773	141	858 GiB	917 GiB	93.55
7	192.168.1.102:9422:/mnt/hd7/	93229	no errors	ok	16 MiB/s	30 MiB/s	183178 us	12681 us	66049 us	2805	122	9	857 GiB	917 GiB	93.48
8	192.168.1.103:9422:/mnt/hd2/	137999	no errors	ok	8.1 MiB/s	9.5 MiB/s	1126301 us	824457 us	328582 us	2225	493	16	1.2 TiB	1.3 TiB	92.02
9	192.168.1.103:9422:/mnt/hd3/	136930	no errors	ok	7.7 MiB/s	15 MiB/s	575956 us	1557387 us	584695 us	1122	4779	101	1.2 TiB	1.3 TiB	92.00
10	192.168.1.103:9422:/mnt/hd4/	137216	no errors	ok	4.7 MiB/s	16 MiB/s	1219459 us	797168 us	393812 us	1651	1215	37	1.2 TiB	1.3 TiB	92.03
11	192.168.1.103:9422:/mnt/hd5/	137669	no errors	ok	5.0 MiB/s	5.0 MiB/s	1105971 us	835725 us	562430 us	1143	1625	229	1.2 TiB	1.3 TiB	92.09
12	192.168.1.103:9422:/mnt/hd6/	137921	no errors	ok	8.1 MiB/s	22 MiB/s	843912 us	48629 us	668692 us	1308	1717	34	1.2 TiB	1.3 TiB	92.10
13	192.168.1.103:9422:/mnt/hd7/	137905	no errors	ok	6.4 MiB/s	10 MiB/s	1186161 us	41384 us	482938 us	921	389	29	1.2 TiB	1.3 TiB	92.01
14	192.168.1.103:9422:/mnt/hd8/	137092	no errors	ok	5.2 MiB/s	34 MiB/s	1902024 us	86157 us	200100 us	875	707	24	1.2 TiB	1.3 TiB	92.06
15	192.168.1.105:9422:/mnt/hd2/	118844	no errors	ok	6.8 MiB/s	25 MiB/s	673062 us	288734 us	756420 us	1554	2342	21	1.2 TiB	1.3 TiB	92.30
16	192.168.1.105:9422:/mnt/hd3/	121137	no errors	ok	3.9 MiB/s	29 MiB/s	691299 us	611552 us	688611 us	743	2446	32	1.2 TiB	1.3 TiB	92.27
17	192.168.1.105:9422:/mnt/hd4/	119424	no errors	ok	5.4 MiB/s	38 MiB/s	506970 us	655564 us	791466 us	744	2857	20	1.2 TiB	1.3 TiB	92.19
18	192.168.1.105:9422:/mnt/hd5/	120183	no errors	ok	7.2 MiB/s	7.1 MiB/s	741746 us	100763 us	728710 us	3449	2104	150	1.2 TiB	1.3 TiB	92.35
19	192.168.1.105:9422:/mnt/hd6/	206214	no errors	ok	4.3 MiB/s	18 MiB/s	752305 us	189387 us	1355969 us	1370	2377	46	1.2 TiB	1.3 TiB	91.73
20	192.168.1.105:9422:/mnt/hd7/	168724	no errors	ok	4.1 MiB/s	8.3 MiB/s	695720 us	174383 us	712481 us	762	3128	356	1.2 TiB	1.3 TiB	91.39
21	192.168.1.106:9422:/mnt/hd2/	142704	no errors	ok	20 MiB/s	37 MiB/s	129437 us	31832 us	242601 us	880	2631	427	1.3 TiB	1.3 TiB	93.41
22	192.168.1.106:9422:/mnt/hd3/	142505	no errors	ok	15 MiB/s	30 MiB/s	181067 us	83917 us	96458 us	2096	1457	147	1.3 TiB	1.3 TiB	93.26
23	192.168.1.106:9422:/mnt/hd4/	143510	no errors	ok	15 MiB/s	15 MiB/s	503939 us	51992 us	959737 us	2604	1899	300	1.3 TiB	1.3 TiB	93.40
24	192.168.1.106:9422:/mnt/hd5/	143824	no errors	ok	12 MiB/s	28 MiB/s	312538 us	63316 us	176195 us	1874	292	7	1.3 TiB	1.3 TiB	93.39
25	192.168.1.106:9422:/mnt/hd6/	141030	no errors	ok	22 MiB/s	31 MiB/s	277476 us	25780 us	66972 us	3376	466	38	1.3 TiB	1.3 TiB	93.52
26	192.168.1.107:9422:/mnt/hd2/	144141	no errors	ok	25 MiB/s	54 MiB/s	169429 us	43006 us	341489 us	2444	1969	56	1.2 TiB	1.3 TiB	92.65
27	192.168.1.107:9422:/mnt/hd3/	86373	no errors	ok	17 MiB/s	37 MiB/s	132352 us	19962 us	163682 us	1862	438	19	858 GiB	917 GiB	93.54

MFS Moose File System

Info + | Servers + | Disks + | Exports − | Mounts + | Operations + | Master Charts + | Server Charts +

W3C css | W3C XHTML 1.0

Exports

#	ip range		path	minversion	alldirs	password	ro/rw	restricted ip	ignore gid	root uid:gid		all users uid:gid	
1	192.168.1.0	192.168.1.255	. (META)	0.0.0	-	no	rw	yes	-	-	-	-	-
2	192.168.1.0	192.168.1.255	/	0.0.0	yes	no	rw	yes	no	0	0	-	-

MFS Moose File System

Info + | Servers + | Disks + | Exports + | Mounts − | Operations + | Master Charts + | Server Charts +

W3C CSS | W3C XHTML 1.0

Active mounts (parameters)

#	session id	host	ip	mount point	version	root dir	ro/rw	restricted ip	ignore gid	root uid:gid		all users uid:gid	
1	62880701	szafa2.local.gem.pl	192.168.1.102	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
2	62880707	szafa3.local.gem.pl	192.168.1.103	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
3	62880743	szafa4.local.gem.pl	192.168.1.104	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
4	62880678	szafa5.local.gem.pl	192.168.1.105	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
5	62880737	szafa6.local.gem.pl	192.168.1.106	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
6	62880658	szafa7.local.gem.pl	192.168.1.107	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
7	62880664	szafa8.local.gem.pl	192.168.1.108	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
8	62880688	szafa8.local.gem.pl	192.168.1.108	/mnt/mfsmeta	1.6.7	. (META)	rw	yes	-	-	-	-	-
9	62880649	szafa9.local.gem.pl	192.168.1.109	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
10	62880650	szafa9.local.gem.pl	192.168.1.109	/mnt/mfsmeta	1.6.7	. (META)	rw	yes	-	-	-	-	-
11	62880740	szafa10.local.gem.pl	192.168.1.110	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
12	62880659	szafa14.local.gem.pl	192.168.1.114	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
13	62880657	szafa15.local.gem.pl	192.168.1.115	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
14	62880667	szafa16.local.gem.pl	192.168.1.116	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
15	62880821	szafa17.local.gem.pl	192.168.1.117	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
16	62885174	szafa18.local.gem.pl	192.168.1.118	/mnt/mfs	1.6.8	/	rw	yes	no	0	0	-	-
17	62883397	szafa19.local.gem.pl	192.168.1.119	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
18	62880886	szafa20.local.gem.pl	192.168.1.120	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
19	62880797	szafa21.local.gem.pl	192.168.1.121	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
20	62880669	szafa23.local.gem.pl	192.168.1.123	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
21	62880818	szafa24.local.gem.pl	192.168.1.124	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
22	62880670	szafa25.local.gem.pl	192.168.1.125	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
23	62880660	szafa26.local.gem.pl	192.168.1.126	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
24	62880828	szafa27.local.gem.pl	192.168.1.127	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
25	62880671	szafa28.local.gem.pl	192.168.1.128	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
26	62880661	szafa29.local.gem.pl	192.168.1.129	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
27	62880662	szafa30.local.gem.pl	192.168.1.130	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
28	62880730	szafa31.local.gem.pl	192.168.1.131	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
29	62880672	szafa32.local.gem.pl	192.168.1.132	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
30	62880839	szafa33.local.gem.pl	192.168.1.133	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-
31	62880785	szafa34.local.gem.pl	192.168.1.134	/mnt/mfs	1.6.7	/	rw	yes	no	0	0	-	-

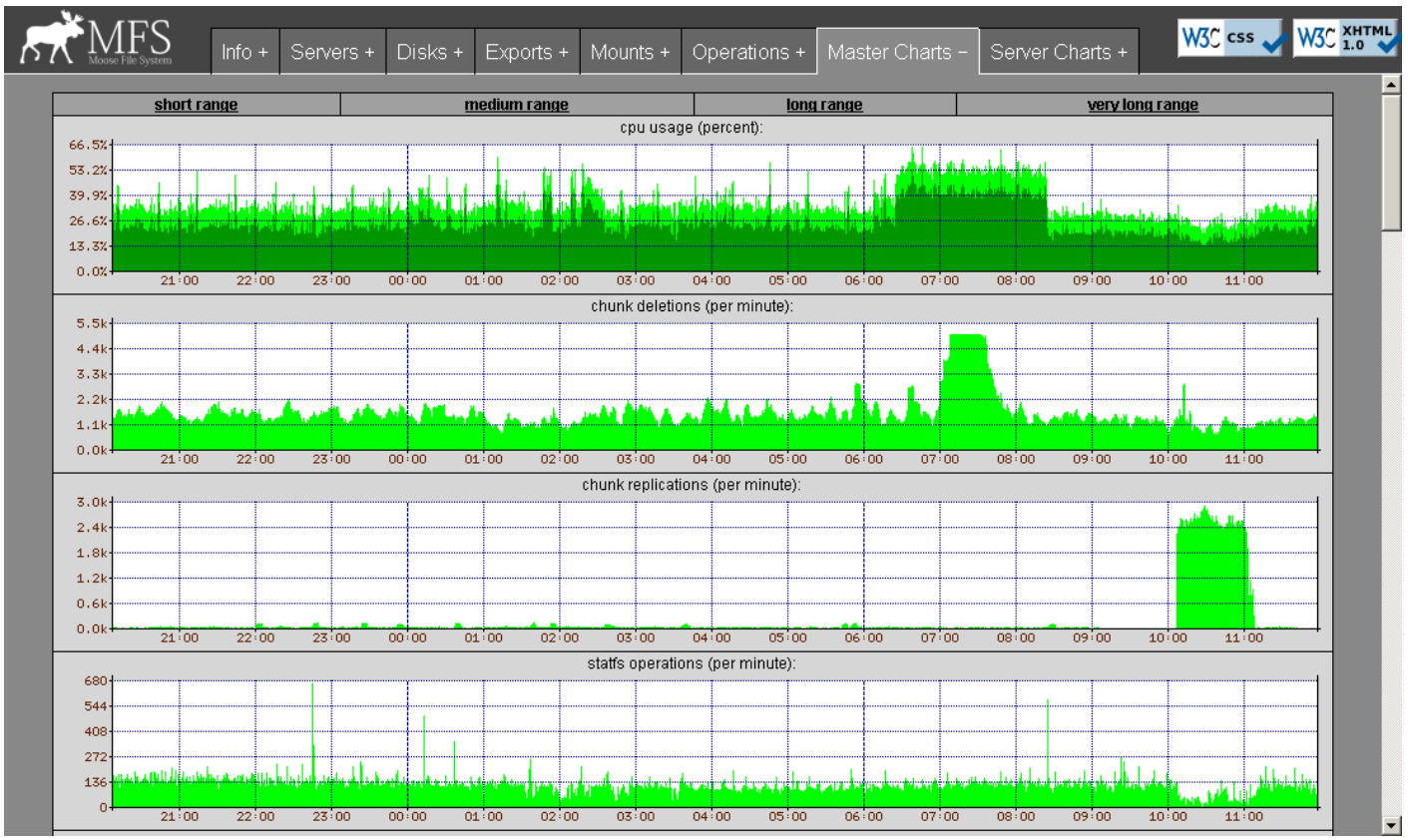
MFS Moose File System

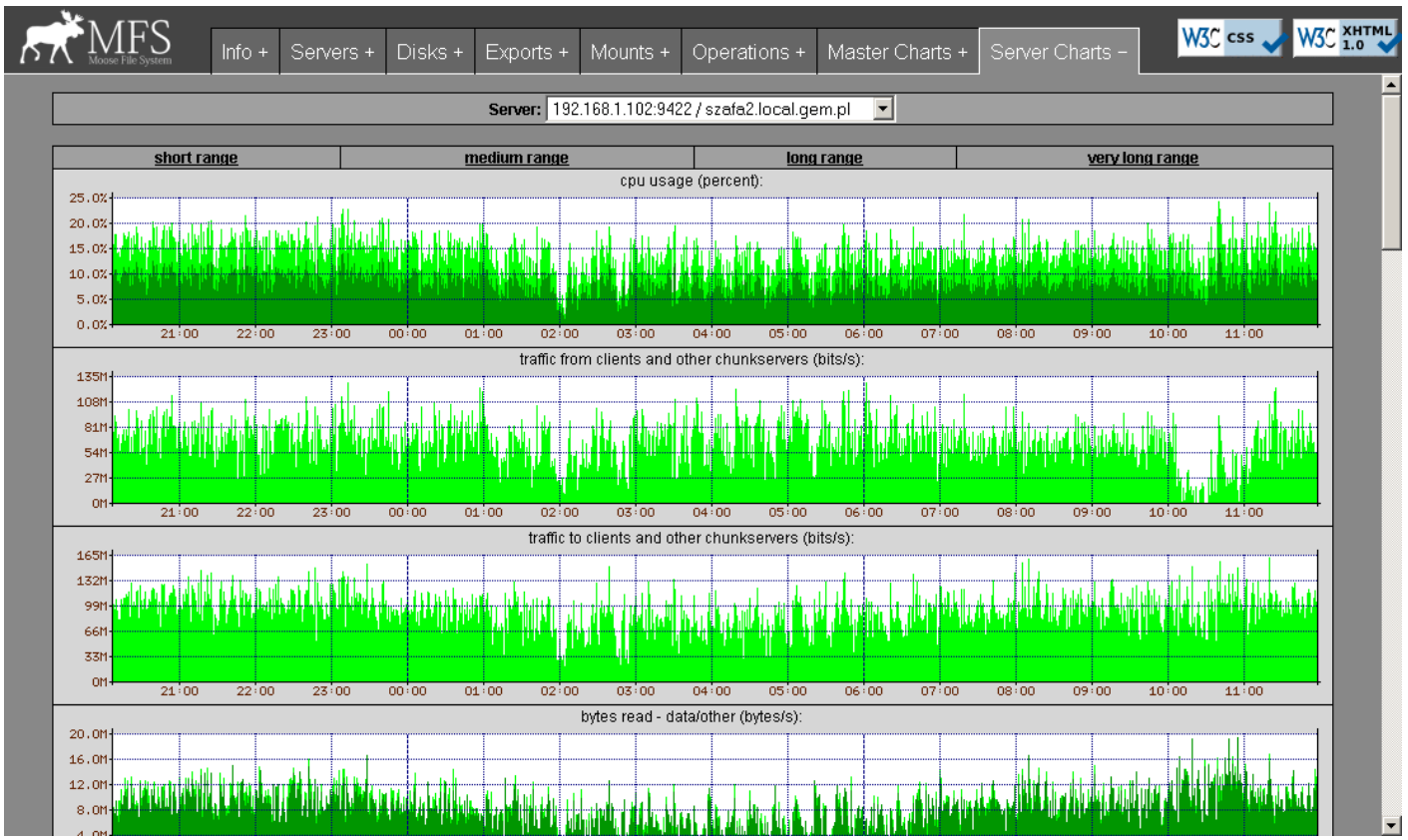
Info + | Servers + | Disks + | Exports + | Mounts + | Operations − | Master Charts + | Server Charts +

W3C CSS | W3C XHTML 1.0

Active mounts (operations)

#	host	ip	mount point	operations current hour/last hour														
				statfs	getattr	setattr	lookup	mkdir	rmdir	symlink	readlink	mknod	unlink	rename	link	readdir	open	read	write
1	szafa2.local.gem.pl	192.168.1.102	/mnt/mfs	60	2486	14	6075	0	0	0	1393	246	80	256	0	267	1294	27087	34116
				40	2274	11	5748	0	0	0	1393	182	62	199	0	271	1136	17568	29763
2	szafa3.local.gem.pl	192.168.1.103	/mnt/mfs	1	13149	334	42013	95	1	0	906	391	1318	347	0	2880	11789	30921	28433
				1	14168	215	45473	72	0	0	73	359	50	225	0	3362	12667	30508	14175
3	szafa4.local.gem.pl	192.168.1.104	/mnt/mfs	8	2438	4782	15732	170	169	0	22	1424	1418	0	0	169	6078	19605	2115
				6	1998	4548	14786	158	158	0	24	1358	1358	0	0	158	5439	17554	2129
4	szafa5.local.gem.pl	192.168.1.105	/mnt/mfs	36	2637	3	3137	0	0	0	0	365	167	356	0	58	948	44816	6902
				24	1726	0	1983	0	0	0	0	206	104	189	0	40	577	37568	6471
5	szafa6.local.gem.pl	192.168.1.106	/mnt/mfs	0	1277	0	4453	0	0	0	1399	0	0	0	0	90	1194	4480	361
				0	1330	0	4493	0	0	0	1414	0	0	0	0	99	1184	4407	365
6	szafa7.local.gem.pl	192.168.1.107	/mnt/mfs	30	2460	5	3076	0	0	0	0	325	151	320	0	57	993	44426	15539
				24	1788	0	2171	0	0	0	0	228	115	211	0	47	673	41618	9583
7	szafa8.local.gem.pl	192.168.1.108	/mnt/mfs	0	368	0	2859	0	0	0	0	0	0	0	0	4	0	0	0
				0	376	0	3006	0	0	0	0	0	0	0	0	3	0	0	0
8	szafa9.local.gem.pl	192.168.1.109	/mnt/mfs	367	23562	952	85683	526	448	0	86	1070	2283	2664	0	19821	17487	17541	27675
				361	25487	821	98027	434	381	0	85	894	1999	2260	0	20992	20830	19346	19690
9	szafa10.local.gem.pl	192.168.1.110	/mnt/mfs	56	1685	8	13667	28	28	0	282	668	114	1537	0	670	1727	1134	7942
				38	1311	8	10225	19	19	0	291	397	99	933	0	661	1421	1149	6001
10	szafa14.local.gem.pl	192.168.1.114	/mnt/mfs	0	897	101	8201	0	0	0	0	216	403	458	0	326	1048	5998	355
				0	724	87	7826	0	0	0	0	188	250	429	0	297	810	1799	326
11	szafa15.local.gem.pl	192.168.1.115	/mnt/mfs	0	958	64	8377	0	0	0	0	120	545	315	0	224	1014	4374	1455
				0	792	51	7858	0	0	0	0	103	394	313	0	205	785	3103	897
12	szafa16.local.gem.pl	192.168.1.116	/mnt/mfs	6	125	540	1628	20	20	0	0	160	162	0	0	20	562	62	236
				6	128	546	1650	20	20	0	0	162	160	0	0	20	567	63	272
13	szafa17.local.gem.pl	192.168.1.117	/mnt/mfs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	szafa18.local.gem.pl	192.168.1.118	/mnt/mfs	0	3043	57	8167	4	0	0	1726	305	0	46	0	4	3884	41678	14914
				1	2121	18	3486	1	0	0	16	281	3	21	0	6	1859	34085	10687
15	szafa19.local.gem.pl	192.168.1.119	/mnt/mfs	2089	21730	0	71457	0	0	0	0	1670	1670	3340	0	1788	4343	2080	7189
				1200	13358	0	50796	0	0	0	0	1450	1450	2900	0	1026	2980	1191	6478


MFS
Moose File System
Info +
Servers +
Disks +
Exports +
Mounts +
Operations +
Master Charts −
Server Charts +
W3C CSS
W3C XHTML 1.0
short range
medium range
long range
very long range
cpu usage (percent):
66.5%
53.2%
39.9%
26.6%
13.3%
0.0%
chunk deletions (per minute):
5.5k
4.4k
3.3k
2.2k
1.1k
0.0k
chunk replications (per minute):
3.0k
2.4k
1.8k
1.2k
0.6k
0.0k
statfs operations (per minute):
680
544
408
272
136
0
21:00
22:00
23:00
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00


MFS
Moose File System
Info +
Servers +
Disks +
Exports +
Mounts +
Operations +
Master Charts +
Server Charts −
W3C CSS
W3C XHTML 1.0
Server: 192.168.1.102:9422 / szafa2.local.gem.pl
short range
medium range
long range
very long range
cpu usage (percent):
25.0%
20.0%
15.0%
10.0%
5.0%
0.0%
traffic from clients and other chunkservers (bits/s):
135M
108M
81M
54M
27M
0M
traffic to clients and other chunkservers (bits/s):
165M
132M
99M
66M
33M
0M
bytes read - data/other (bytes/s):
20.0M
16.0M
12.0M
8.0M
21:00
22:00
23:00
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00

Info − | Servers + | Disks + | Exports + | Mounts + | Operations + | Master Charts + | Server Charts +

Info												
version	total space	avail space	trash space	trash files	reserved space	reserved files	all fs objects	directories	files	chunks	all chunk copies	regular chunk copies
1.6.11	144 GiB	87 GiB	0 B	0	0 B	0	2	1	1	782	782	782

All chunks state matrix (counts 'regular' hdd space and 'marked for removal' hdd space : switch to 'regular')												
goal	valid copies											
	0	1	2	3	4	5	6	7	8	9	10+	all
0	-	-	-	-	-	-	-	-	-	-	-	0
1	-	782	-	-	-	-	-	-	-	-	-	782
2	-	-	-	-	-	-	-	-	-	-	-	0
3	-	-	-	-	-	-	-	-	-	-	-	0
4	-	-	-	-	-	-	-	-	-	-	-	0
5	-	-	-	-	-	-	-	-	-	-	-	0
6	-	-	-	-	-	-	-	-	-	-	-	0
7	-	-	-	-	-	-	-	-	-	-	-	0
8	-	-	-	-	-	-	-	-	-	-	-	0
9	-	-	-	-	-	-	-	-	-	-	-	0
10+	-	-	-	-	-	-	-	-	-	-	-	0
all 1+	0	782	0	0	0	0	0	0	0	0	0	782

Chunk operations info							
loop time		deletions				replications	
start	end	invalid	unused	disk clean	over goal	under goal	rebalance
Thu Jan 14 19:30:44 2010	Thu Jan 14 19:35:43 2010	0/0	0/0	0/0	0/0	0/0	0

Filesystem check info							
check loop start time	check loop end time	files	under-goal files	missing files	chunks	under-goal chunks	missing chunks
no data							

Info + | Servers − | Disks − | Exports − | Mounts − | Operations − | Master Charts + | Server Charts +

Chunk Servers													
#	host	ip	port	version	'regular' hdd space				'marked for removal' hdd space			
					chunks	used	total	% used	chunks	used	total	% used
1	(unresolved)	192.168.28.213	9422	1.6.11	260	19 GiB	48 GiB	39.82	0	0 B	0 B	-
2	(unresolved)	192.168.28.231	9422	1.6.11	261	19 GiB	48 GiB	39.85	0	0 B	0 B	-
3	(unresolved)	192.168.28.232	9422	1.6.11	261	19 GiB	48 GiB	39.95	0	0 B	0 B	-

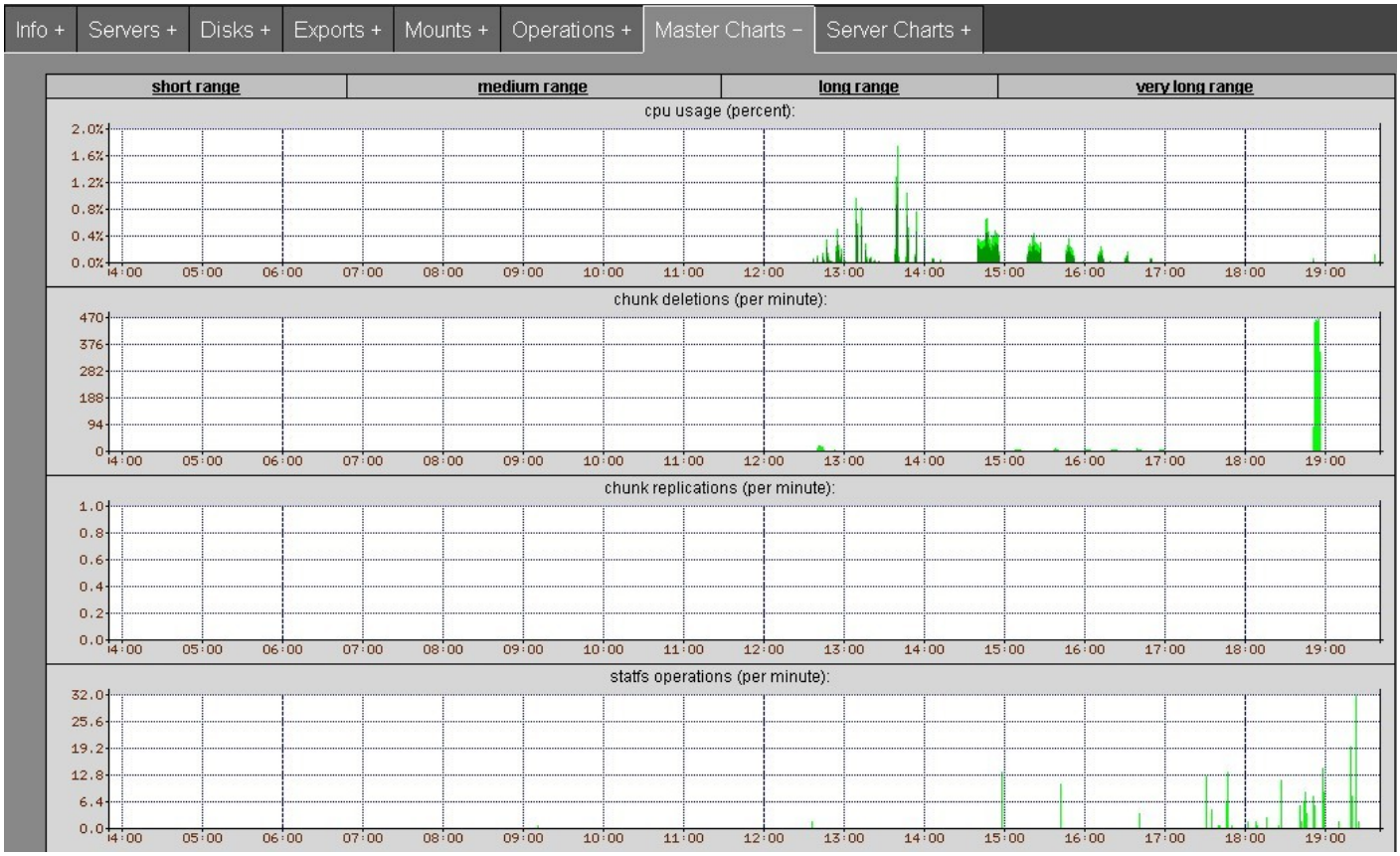
Metadata Backup Loggers			
#	host	ip	version

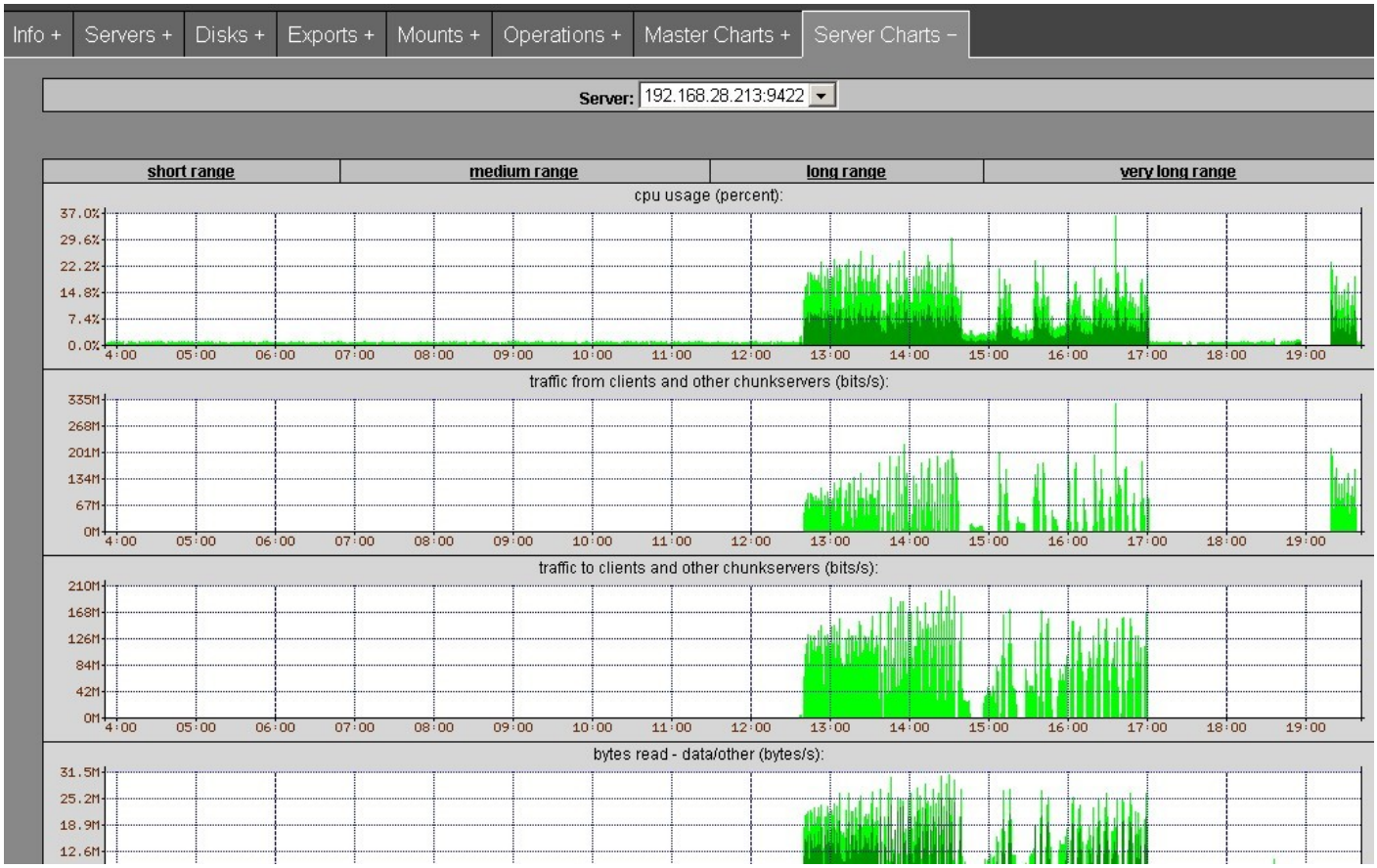
Disks																
#	info				I/O stats last min (switch to hour,day)								space		
					transfer		max time (switch to avg)			# of ops					
	path	chunks	last error	status	read	write	read	write	fsync	read	write	fsync	used	total	used (%)
1	192.168.28.213:9422:/data/	260	no errors	ok	-	-	-	-	-	-	-	-	19 GiB	48 GiB	39.82
2	192.168.28.231:9422:/data/	261	no errors	ok	-	-	-	-	-	-	-	-	19 GiB	48 GiB	39.85
3	192.168.28.232:9422:/data/	261	no errors	ok	-	-	-	-	-	-	-	-	19 GiB	48 GiB	39.95

Exports												
#	ip range		path	minversion	alldirs	password	ro/rw	restricted ip	ignore gid	root uid:gid		all users uid:gid	
1	192.168.28.0	192.168.28.255	. (META)	0.0.0	-	no	rw	yes	-	-	-	-	-
2	192.168.28.0	192.168.28.255	/	0.0.0	no	no	rw	yes	no	999	999	-	-

Active mounts (parameters)													
#	session id	host	ip	mount point	version	root dir	ro/rw	restricted ip	ignore gid	root uid:gid		all users uid:gid	
1	2	(unresolved)	192.168.28.233	/mnt/mfsmeta	1.6.11	. (META)	rw	yes	-	-	-	-	-
2	3	(unresolved)	192.168.28.233	/mnt/mfs	1.6.11	/	rw	yes	no	999	999	-	-

Active mounts (operations)																		
#	host	ip	mount point	operations current hour/last hour														
				statfs	getattr	setattr	lookup	mkdir	rmdir	symlink	readlink	mknod	unlink	rename	link	readdir	open	read	write
1	(unresolved)	192.168.28.233	/mnt/mfs	32	23	0	9	0	2	0	0	1	0	0	0	18	1	0	965
				50	36	0	10	2	0	0	0	0	0	0	0	8	0	0	0


Info +
Servers +
Disks +
Exports +
Mounts +
Operations +
Master Charts −
Server Charts +
short range
medium range
long range
very long range
cpu usage (percent):
chunk deletions (per minute):
chunk replications (per minute):
statfs operations (per minute):


Info +
Servers +
Disks +
Exports +
Mounts +
Operations +
Master Charts +
Server Charts −
Server: 192.168.28.213:9422
short range
medium range
long range
very long range
cpu usage (percent):
traffic from clients and other chunkservers (bits/s):
traffic to clients and other chunkservers (bits/s):
bytes read - data/other (bytes/s):

1.7 常见问题及建议对策

1.7.1 Master 性能瓶颈

master 本身的性能瓶颈。不太恰当的比方：类似 mysql 主从复制，从的可以扩展，主的不容易扩展。（qq 群战友 ： hzqbbc ）

短期对策：按业务切分

1.7.2 体系架构存储文件总数的瓶颈。

mfs 把文件系统的结构缓存到 master 的内存中，个人认为文件越多，master 的内存消耗越大，8g 对应 2500kw 的文件数，2 亿文件就得 64GB 内存。（qq 群战友 ： hzqbbc ）

短期对策：按业务切分

1.7.3 单点故障解决方案的健壮性。（qq 群战友 ： tt ，hzqbbc）

1.7.4 垃圾回收

默认的垃圾回收时间是 86400，存在一种可能性是垃圾还没回收完，你的存储容量就暴掉了。（案例提供者 shinelian）

方案一：设置垃圾回收时间，积极监控存储容量。

经过测试，把垃圾回收时间设置 300 秒，完全可以正确回收容量。

方案二：手动周期性去删除 metamfs 里的 trash 目录下的文件（健壮性还有待测试，反正删除后容量是回收了，不晓得有没有什么后遗症。）

经过测试，貌似没后遗症，有后遗症的同学请在 qq 群里面联系我。

2 安装

硬件环境：

Master Server　1 台

Chunk Server 3 台

client　1 台

操作系统：

CentOS 5.3 X64

2.1 MooseFS Master 的安装

2.1.1 安装

```
wget http://ncu.dl.sourceforge.net/project/moosefs/moosefs/1.6.11/mfs-1.6.11.tar.gz
tar zxvf mfs-1.6.11.tar.gz
cd mfs-1.6.11
useradd mfs -s /sbin/nologin
```

```
./configure --prefix=/usr/local/mfs --with-default-user=mfs --with-default-group=mfs
make
make install
cd /usr/local/mfs/etc/
cp mfsmaster.cfg.dist mfsmaster.cfg
cp mfsexports.cfg.dist mfsexports.cfg
vim mfsmaster.cfg
vim mfsexports.cfg
cd ..
cd var/
mfs/
cp metadata.mfs.empty metadata.mfs
cat metadata.mfs
/usr/local/mfs/sbin/mfsmaster start
ps axu | grep mfsmaster
lsof -i
tail -f /var/log/messages
```

2.1.2 启动Master服务

```
/usr/local/mfs/sbin/mfsmaster start
working directory: /usr/local/mfs/var/mfs
lockfile created and locked
initializing mfsmaster modules ...
loading sessions ... ok
sessions file has been loaded
exports file has been loaded
loading metadata ...
create new empty filesystemmetadata file has been loaded
no charts data file - initializing empty charts
master <-> metaloggers module: listen on *:9419
master <-> chunkservers module: listen on *:9420
main master server module: listen on *:9421
mfsmaster daemon initialized properly
```

2.1.3 停止Master服务

```
/usr/local/mfs/sbin/mfsmaster -s
```

2.1.4 启动和停止Web GUI

```
启动： /usr/local/mfs/sbin/mfscgiserv
停止： kill /usr/local/mfs/sbin/mfscgiserv
```

2.1.5 相关配置文件

```
vim mfsexports.cfg
192.168.28.0/24 . rw
```

```
192.168.28.0/24 /    rw
```

2.2 MooseFS Chunk Server 的安装

2.2.1 从块设备创建本地文件系统

```
fdisk -l
mkfs.ext3 /dev/sdb
mkdir /data
chown mfs:mfs /data
mount -t ext3 /dev/sdb /data
df -ah
/dev/sdb       133G  188M  126G  1% /data
```

2.2.2 创建 50G 的 Loop Device 文件

```
df -ah
dd if=/dev/zero of=/opt/mfs.img bs=1M count=50000
losetup /dev/loop0 mfs.img
mkfs.ext3 /dev/loop0
mkdir /data
chown mfs:mfs /data
mount -o loop /dev/loop0 /data
df -ah
```

2.2.3 安装 Chunk Server

```
wget http://ncu.dl.sourceforge.net/project/moosefs/moosefs/1.6.11/mfs-1.6.11.tar.gz
tar zxvf mfs-1.6.11.tar.gz
cd mfs-1.6.11
useradd mfs -s /sbin/nologin
./configure --prefix=/usr/local/mfs --with-default-user=mfs --with-default-group=mfs
make
make install
cd /usr/local/mfs/etc/
cp mfschunkserver.cfg.dist mfschunkserver.cfg
cp mfshdd.cfg.dist mfshdd.cfg
```

2.2.4 启动 Chunk Server

```
/usr/local/mfs/sbin/mfschunkserver start
ps axu |grep mfs
tail -f /var/log/messages
```

2.2.5 停止 Chunk Server

```
/usr/local/mfs/sbin/mfschunkserver stop
```

2.3 MooseFS Client 的安装

2.3.1 安装 fuse

```
yum install kernel.x86_64 kernel-devel.x86_64 kernel-headers.x86_64
###reboot server####
yum install fuse.x86_64 fuse-devel.x86_64 fuse-libs.x86_64
modprobe fuse
```

2.3.2 安装 MooseFS Client

```
wget http://ncu.dl.sourceforge.net/project/moosefs/moosefs/1.6.11/mfs-1.6.11.tar.gz
tar zxvf mfs-1.6.11.tar.gz
cd mfs-1.6.11
useradd mfs -s /sbin/nologin
./configure --prefix=/usr/local/mfs --with-default-user=mfs --with-default-group=mfs
--enable-mfsmount
make
make install
```

2.3.3 挂载文件系统

```
cd /mnt/
mkdir mfs
/usr/local/mfs/bin/mfsmount /mnt/mfs/ -H 192.168.28.242

mkdir mfsmeta
/usr/local/mfs/bin/mfsmount -m /mnt/mfsmeta/ -H 192.168.28.242

df -ah
```

3 系统管理

3.1 管理命令

设置副本 的份数，推荐 3 份

```
/usr/local/mfs/bin/mfssetgoal -r 3 /mnt/mfs
```

查看某文件

```
/usr/local/mfs/bin/mfsgetgoal /mnt/mfs
```

查看目录信息

```
/usr/local/mfs/bin/mfsdirinfo -H /mnt/mfs
```

4 性能测试

4.1 MooseFS

4.1.1 大文件

block=1M byte

```
dd if=/dev/zero of=1.img bs=1M count=5000
5242880000 bytes (5.2 GB) copied, 48.8481 seconds, 107 MB/s
```

4.1.2 小文件测试一

50 byte * 100w个 * 1 client

1. 写入(1000 * 1000)

```
real  83m41.343s
user  4m17.993s
sys  16m58.939s
```

2. 列表

```
time find ./ -type f | nl | tail
999999 ./0/1
1000000 ./0/0
real  0m39.418s
user  0m0.721s
sys  0m0.225s
```

3. 删除

```
time rm -fr *
real  6m35.273s
user  0m0.394s
sys  0m23.546s
```

4.1.3 小文件测试二

1K byte * 100w个 * 100 client 1000 * 1000

1. 写入 (100 Client)

```
time ../../p_touch_file.sh
real  22m51.159s
user  4m42.850s
sys  18m41.437s
```

2. 列表(1 Client)

```
time find ./ | nl | tail
real  0m35.910s
user  0m0.628s
sys  0m0.204s
```

3. 删除 (1 Client)

```
time rm -fr *
real  6m36.530s
user  0m0.697s
sys  0m21.682s
```

4.1.4 小文件测试三

1k byte* 100w 个 * 200 client 1000 * 1000

```
time ../../p_touch_file.sh
real  27m56.656s
user  5m12.195s
sys  20m52.079s
```

4.1.5 小文件测试四

1k byte* 100w 个 * 1000 client 1000 * 1000

```
time ../../p_touch_file.sh
real  30m30.336s
user  5m6.607s
sys  21m
```

4.2 本地磁盘

4.2.1 大文件

block=1M byte

```
dd if=/dev/zero of=1.img bs=1M count=5000
5242880000 bytes (5.2 GB) copied, 58.7371 seconds, 89.3 MB/s
```

4.2.2 小文件

50 byte * 100w 个 * 1 client 1000 * 1000

1. 写入

```
time ../touch_file.sh
real 17m47.746s
user 4m54.068s
sys 12m54.425s
```

2. 列表

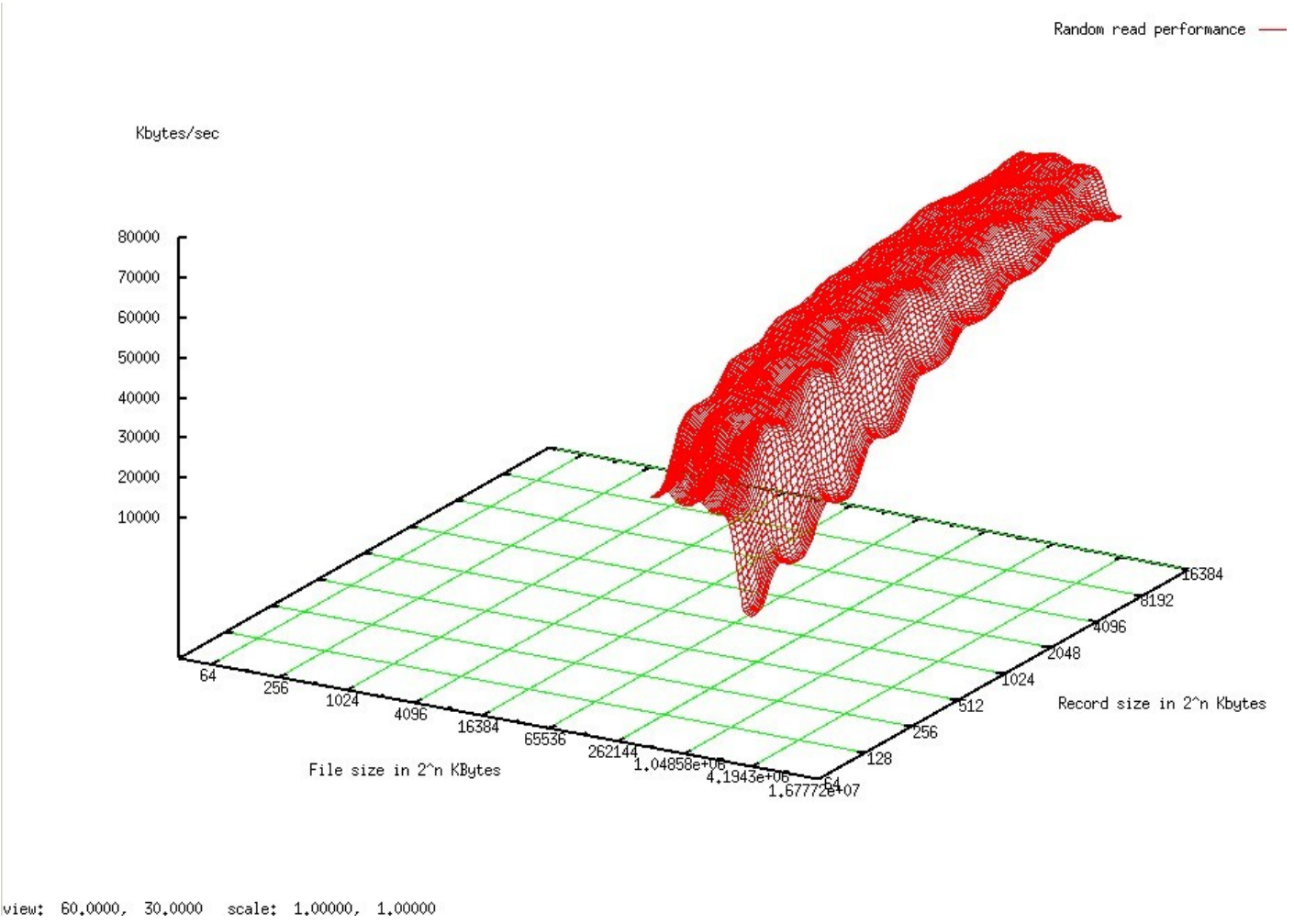
```
time find ./ -type f | nl | tail
1000000 ./875/582
1000001 ./875/875
real 0m9.120s
user 0m1.102s
sys 0m0.726s
```

3. 删除

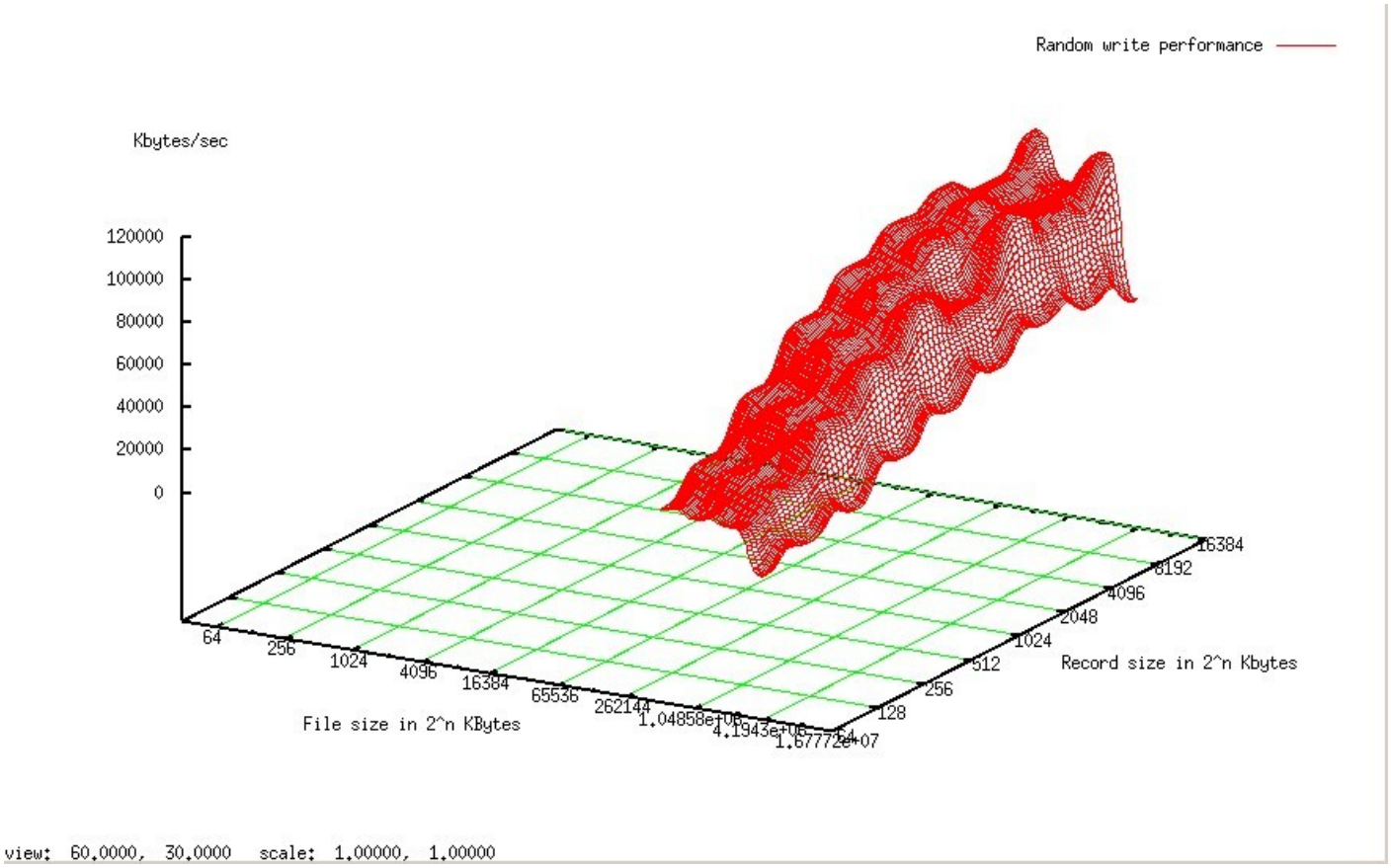
```
time rm -fr *
real 0m37.201s
user 0m0.432s
sys 0m15.268s
```

4.3 基准测试（第一次）

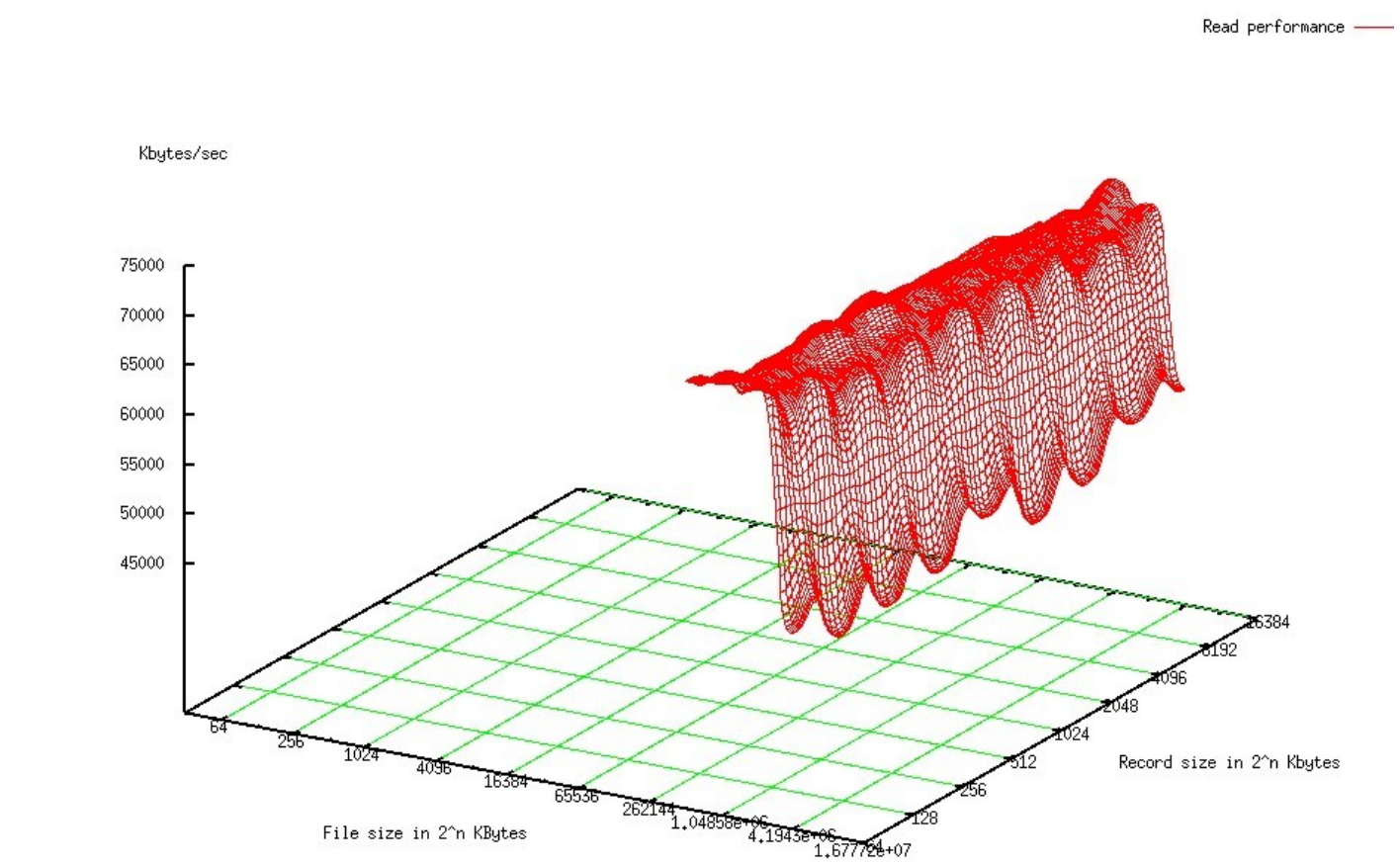
4.3.1 随机读



4.3.2 随机写

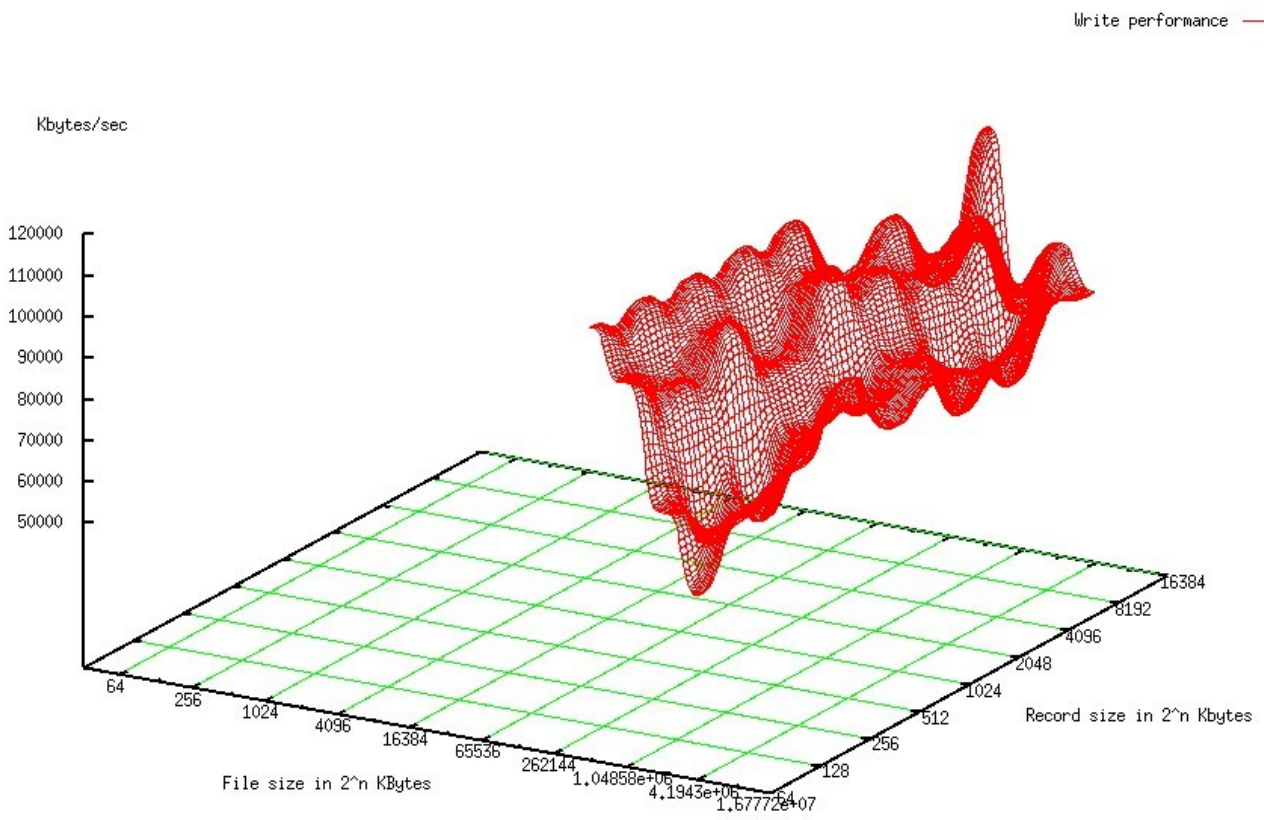


4.3.3 顺序读



view: 60.0000, 30.0000 scale: 1.00000, 1.00000

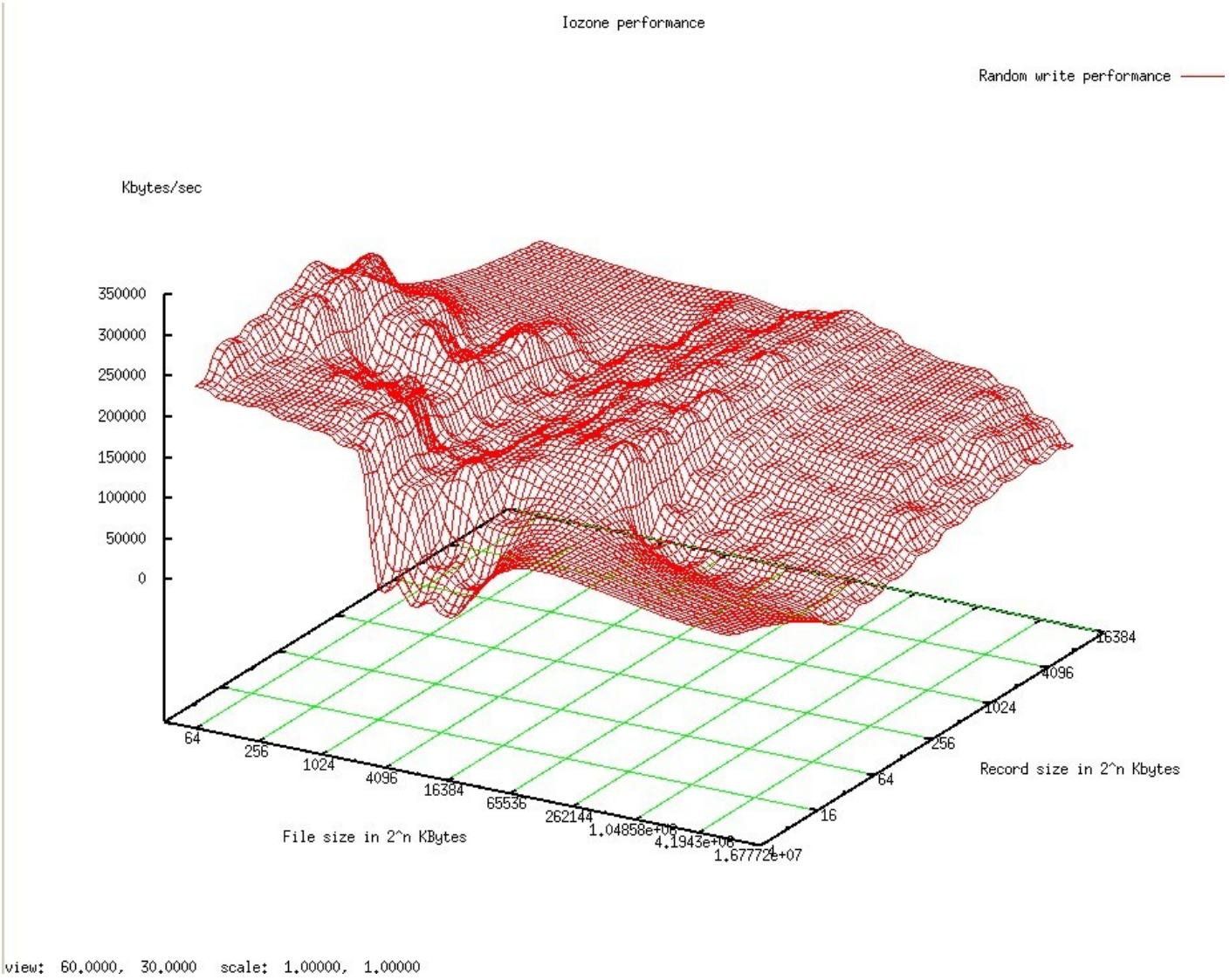
4.3.4 顺序写



view: 60.0000, 30.0000 scale: 1.00000, 1.00000

4.4 基准测试（第二次）

4.4.1 随机读



5 参考文献

5.1 文献

http://sery.blog.51cto.com/10037/263515 田逸

http://bbs.chinaunix.net/thread-1643863-1-1.html ltgzs777

http://www.moosefs.org/ 官网

http://bbs.chinaunix.net/thread-1643015-1-2.html 测试工具

5.2 测试数据

5.2.1 性能测试模型一

一个不知道名字的哥们的测试结果，我先贴出来，那哥们看到了密我。

小文件性能测试			
二级 100*100 文件夹	创建	列表	删除

单片 15k.5 ext3 client 单进程[2]	Real 0m0.762s user 0m0.048s sys 0m0.261s	Real 0m0.179s user 0m0.036s sys 0m0.125s	Real 0m0.492s user 0m0.036s sys 0m0.456s
单片 15k.5 ext3 client 10 并发进程[3]	最长时间： real 0m0.724s user 0m0.015s sys 0m0.123s	最长时间： real 0m0.057s user 0m0.006s sys 0m0.025s	最长时间： real 0m0.226s user 0m0.010s sys 0m0.070s
6chunkserver cache client 单进程	Real 0m2.084s user 0m0.036s sys 0m0.252s	Real 0m4.964s user 0m0.043s sys 0m0.615s	Real 0m6.661s user 0m0.046s sys 0m0.868s
6chunkserver cache client 10 并发进程	最长时间： real 0m1.422s user 0m0.007s sys 0m0.050s	最长时间： real 0m2.022s user 0m0.008s sys 0m0.108s	最长时间： real 0m2.318s user 0m0.008s sys 0m0.136s
二级 1000*1000 文件夹	**创建**	**列表**	**删除**
单片 15k.5 ext3 client 单进程	Real 11m37.531s user 0m4.363s sys 0m37.362s	Real 39m56.940s user 0m9.277s sys 0m48.261s	Real 41m57.803s user 0m10.453s sys 3m11.808s
单片 15k.5 ext3 client 10 并发进程	最长时间： real 11m7.703s user 0m0.519s sys 0m10.616s	最长时间： real 39m30.678s user 0m1.031s sys 0m4.962s	最长时间： real 40m23.018s user 0m1.043s sys 0m19.618s
6chunkserver cache client 单进程	Real 3m17.913s user 0m3.268s sys 0m30.192s	Real 11m56.645s user 0m3.810s sys 1m10.387s	Real 12m14.900s user 0m3.799s sys 1m26.632s
6chunkserver cache client 10 并发进程	最长时间： real 1m13.666s user 0m0.328s sys 0m3.295s	最长时间： real 4m31.761s user 0m0.531s sys 0m10.235s	最长时间： real 4m26.962s user 0m0.663s sys 0m13.117s
三级 100*100*100 文件夹	**创建**	**列表**	**删除**
单片 15k.5 ext3 client 单进程	Real 9m51.331s user 0m4.036s sys 0m32.597s	Real 27m24.615s user 0m8.907s sys 0m44.240s	Real 28m17.194s user 0m10.644s sys 1m34.998s
单片 15k.5 ext3 client 10 进程	最长时间： real 10m22.170s user 0m0.580s sys 0m11.720s	最长时间： real 33m32.386s user 0m1.127s sys 0m5.280s	最长时间： real 33m7.808s user 0m1.196s sys 0m10.588s
6chunkserver cache	Real 3m21.720s	Real 9m26.535s	Real 10m51.558s

2 单盘多进程性能没有提升，因为都在 io wait，甚至增加进程会消耗大量调度时间

3 MFS 多进程时性能会提升，主要性能消耗集中在 CPU 系统时间。因此实际海量小文件性能要大大强于本地文件系统

client 单进程	user 0m3.089s sys 0m26.635s	user 0m3.901s sys 1m11.756s	user 0m4.186s sys 1m26.322s
6chunkserver cache client 10 并发进程	最长时间： real 1m23.023s user 0m0.429s sys 0m3.869s	最长时间： real 4m10.617s user 0m0.643s sys 0m11.588s	最长时间： real 4m20.137s user 0m0.649s sys 0m14.120s
6chunkserver cache client 50 并发进程	最长时间： real 1m26.388s user 0m0.074s sys 0m0.679s	最长时间： real 4m37.102s user 0m0.132s sys 0m2.160s	最长时间： real 4m37.392s user 0m0.132s sys 0m2.755s
6chunkserver cache client 100 并发进程	最长时间： real 1m29.338s user 0m0.062s sys 0m0.363s	最长时间： real 4m54.925s user 0m0.069s sys 0m1.212s	最长时间： real 4m35.845s user 0m0.068s sys 0m1.640s
6chunkserver cache remote client 10 并发进程	最长时间： real 4m0.411s user 0m2.985s sys 0m12.287s	最长时间： real 8m31.351s user 0m4.223s sys 0m29.800s	最长时间： real 4m3.271s user 0m3.206s sys 0m11.922s

三级 100*100*100 文件夹	**1**	**2**	**3**	**4**	**5**
变更日志/元数据大小	55M 左右	60M 左右	60M 左右	60M 左右	60M 左右
连续创建耗时	Real 4m0.411s user 0m2.985s sys 0m12.287s	Real 4m12.309s user 0m3.039s sys 0m12.899s	Real 4m14.010s user 0m3.418s sys 0m12.831s	Real 4m14.214s user 0m3.247s sys 0m12.871s	Real 4m14.417s user 0m3.170s sys 0m12.948s

5.2.2 性能测试模型二[4]

两个 Client 同时 dd 测试

数据块 1M 文件大小 20G

Client1 写：68.4MB/s 读：25.3MB/s

Client2 写：67.5MB/s 读：24.7MB/s

总吞吐：写：135.9MB/s 读：50.0MB/s

写命令：dd if=/dev/zero of=/mfs/test.1 bs=1M count=20000

读命令：dd if=/mfs/test.1 of=/dev/null bs=1M

4 感谢 qq 群战友 痞子白 提供

6 MooseFS 1.5.x 数据恢复实例

其实很简单，就是 mfsrestore，然后启动服务的时候，并有任何提示信息，进程启不来,其实是配置文件放 PID 的目录会被删掉，重建这个目录，然后赋予权限就可以了,我已经做过 1.5 到 1.6 的升级，还可以。

详情见 Xufeng blog http://snipt.net/iamacnhero/tag/moosefs

7 MooseFS 热备方案

前 2 天问了 mfs 作者关于 性能测试和灾难恢复这 2 块的情况，如下是回复，随信还附了 mini how to。

Hi!

We are very happy that you are a user of MooseFS! Also thank you very much
for promoting the system in China!

Regarding the Distaster Recovery please read the mini how which I attach -
you can easily change the metalogger machine to be the master server.

Regarding performance tests - we have not done any extensive tests. And we
are happy that people from open source community like you contribute to it
and fill the gap.

We are preparing our new website and we would like to put there
descriptions of installed MooseFS systems all over the world. Could you
write more about your architecture? What is your master server? How many
chunkservers do you have? What is the total space used by you? And so on
and so on - the more details, the better.

Kind regards
Michał Borychowski
MooseFS Support Manager

_ _
Gemius S.A.
ul. Wołoska 7, 02-672 Warszawa
Budynek MARS, klatka D
Tel.: +4822 874-41-00;
Fax : +4822 874-41-01

##3

How to prepare a fail proof solution with a redundant master?

A redundant master functionality is right now not a built-in functionality. But this subject is for us very crucial because we know how important this is and we receive lots of requests about it from many sources.

master 的冗余能力目前还不是一个原生或者内建的功能，但这对我们来说是一个非常关键的课题，因为它是非常重要以及近来从许多地方受到了大量关于此问题的询问。

It is important to mention that even in MooseFS v 1.5.x edition it is relatively easy to write a scripts set which would quite automatically start a backup master server and in 1.6.x it is even simpler. The whole process of switching to the backup server would take less than a minute.

特别重要的，在 moosefe1.5.x 系列已经可以通过编写脚本来自动的启动一个备援的 master，在 1.6.x 系列则更为简单。切换到备援 master 的整个过程会花掉不到一分钟的时间。

It is enough to use for example Common Address Redundancy Protocol (http://www.openbsd.org/faq/pf/carp.html, http://en.wikipedia.org/wiki/Common_Address_Redundancy_Protocol). CARP allows that there exist two machines with the same IP in one LAN - one MASTER and the second BACKUP.

So you can set up IP of mfsmaster on a CARP interface and configure the master machine to be used as MooseFS main master. On the backup machine you also install mfsmaster but of course you do not run it.

Versions 1.6.5 and above contain a new program "mfsmetalogger" which you can run on whatever machine you wish. The program gets metadata from master - every several hours (by default every 24 hours) gets a full metadata file and on current basis a complete change log.

If you run an earlier version of MooseFS than 1.6.5 it is enough to set up several simple scripts run regularly from cron (eg. every one hour) which would backup metadata file from the main master "PREFIX/var/mfs/metadata.mfs.back".

You also need an extra script run continuously testing the CARP interface state which in case if this interface goes in a MASTER mode would get two or three newest "changelog" files from any chunkserver (just by using "scp"), would also start "mfsmetarestore" and then "mfsmaster". The switch time should take approximately several seconds and along with time necessary for

reconnecting chunkservers a new master would be fully functional in about a minute (in both read and write modes).

We also plan to add option to run master in read-only mode - which would be the best option for running the backup machine. This would secure the system against potential desynchronization of the two master machines and the need of merging all the changes to the main master which took place on the backup master.

8 附录

8.1 1000 * 1000 * 1 client 脚本

```
#!/bin/bash
for ((i=0;i<1000;i++))
do
  mkdir ${i}
  cd ${i}
  for ((j=0;j<1000;j++))
   do
    cp /mnt/test ${j}
   done
   cd ..
done
```

8.2 1000 * 1000 * (100 , 200 ,1000 client) 脚本

```
#!/bin/bash
declare -f make_1000_dir_file
cd `pwd`
function make_1000_dir_file {
  start=${1}
  stop=${2}
  for ((i=${start};i<${stop};i++))
  do
    mkdir ${i}
    for ((j=0;j<1000;j++))
    do
      cp /mnt/test ${i}/${j}
    done
  done
}
i=1
while [ ${i} -le 1000 ]
do
  ((n=${i}+1))
```

```
  make_1000_dir_file ${i} $ &
  ((i=${i}+1))
done
wait
```

8.3 mfs 官方关于 1.6.x 的介绍[5]

View on new features of next release v 1.6 of Moose File System

关于对 MFS(Moose File System)下一个发布版本 V1.6 新特性的一些看法

We are about to release a new version of MooseFS which would include a large number of new features and bug fixes. The new features are so significant that we decided to release it under 1.6 version. The newest beta files are in the GIT repository.

我们将要发布 MFS 一个最新版本，该版本修复了大量的 bug,同时也包含了大量的新特性。这些新特性非常重要和有特色，我们决定在 1.6 版本进行发布。最新的 beta 文件你可以 GIT 的知识库找得到。

The key new features/changes of MooseFS 1.6 would include:

MooseFS 1.6 的主要特性及变化包括：

8.3.1 General

Removed duplicate source files.

移除了复制源文件

Strip whitespace at the end of configuration file lines.

配置文件行的末尾将为空白

8.3.2 Chunkserver

Rewritten in multi-threaded model.

重写了多线成模式

Added periodical chunk testing functionality (HDD_TEST_FREQ option).

增加了定期 chunk 测试功能（HDD_TEST_FREQ 选项）

New -v option (prints version and exits).

新的-v 选项（显示版本）

8.3.3 Master

Added "noowner" objects flag (causes objects to belong to current user).

增加了"noowner"对象标记（可以使对象属于当前用户）

Maintaining `mfsdirinfo` data online, so it doesn't need to be calculated on every

---

5 译者：QQ 群战友：Cuatre

request.

保持‘mfsdirinfo’数据在线，这样就不需要求每一个请求都进行运算。

Filesystem access authorization system (NFS-like mfsexports.cfg file, REJECT_OLD_CLIENTS option) with ro/rw, maproot, mapall and password functionality.

文件系统访问认证系统（类似于 NFS 的 mfsexports.cfg 文件，REJECT_OLD_CLIENTS 选项），有 ro/rw, maproot, mapall 及密码功能

New -v option (prints version and exits).

新的 -v 选项（显示版本）

8.3.4 Mount

Rewritten options parsing in mount-like way, making possible to use standard FUSE mount utilities (see mfsmount(manual for new syntax). Note: old syntax is no longer accepted and mountpoint is mandatory now (there is no default).

重写选项将采用类似于挂载的解析方式，使用标准的 FUSE 挂载工具集将成为可能（参见新的 mfsmount(语法手册）。注：旧的语法现在将不再被支持，而设置挂载点则是必须的。（非默认选项）

Updated for FUSE 2.6+.

升级到 FUSE 2.6 版本以上

Added password, file data cache, attribute cache and entry cache options. By default attribute cache and directory entry cache are enabled, file data cache and file entry cache are disabled.

增加了密码，文件数据缓存，属性缓存及目录项选项。默认情况下，属性缓存及目录项缓存是开启的，而文件数据缓存和文件项输入缓存则是关闭的

opendir() no longer reads directory contents- it's done on first readdir() now; fixes "rm -r" on recent Linux/glibc/coreutils combo.

opendir()函数将不再读取目录内容-读取目录内容现在将由 readdir()函数完成；修复了当前 Linux/glibc/coreutils 组合中的‘rm -r’命令

Fixed mtime setting just before close() (by flushing file on mtime change); fixes mtime preserving on "cp -p".

修复了在 close()前的 mtime 设置（在 mtime 变化的时候刷新文件）

Added statistics accessible through MFSROOT/.stats pseudo-file.

增加了表示访问吞吐量的统计伪文件 MFSROOT/.stats

Changed master access method for mfstools (direct .master pseudo-file replaced by .masterinfo redirection); fixes possible mfstools race condition and allows to use mfstools on read-only filesystem.

对于 mfstools 改变了主要的访问路径（直接）

8.3.5 Tools

Units cleanup in values display (exact values, IEC-60027/binary prefixes, SI/decimal prefixes); new options: -n, -h, -H and MFSHRFORMAT environment variable - refer to mfstools(manual for details).

在单元值显示方面进行一致化（确切值，IEC-60027/二进制前缀， SI/十进制前缀）；新的选项：-n,-h,-H以及可变的 MFSHRFORMAT 环境----详细参见 mfstools(手册

mfsrgetgoal, mfsrsetgoal, mfsrgettrashtime, mfsrsettrashtime have been deprecated in favour of new "-r" option for mfsgetgoal, mfssetgoal, mfsgettrashtime, mfssettrashtime tools.

我们推荐使用带新的“-r”选项的 mfsgetgoal, mfssetgoal, mfsgettrashtime, mfssettrashtime 工具，而不推荐 mfsrgetgoal, mfsrsetgoal, mfsrgettrashtime, mfsrsettrashtime 工具。（注意前后命令是不一样的，看起来很类似）

mfssnapshot utility replaced by mfsappendchunks (direct descendant of old utility) and mfsmakesnapshot (which creates "real" recursive snapshots and behaves similar to "cp -r".

mfssnapshot 工具集取代了 mfsappendchunks（老工具集的后续版本）和 mfsmakesnapshot（该工具能够创建“真”的递归快照，这个动作类似于执行“cp -r”）工具

New mfsfilerepair utility, which allows partial recovery of file with some missing or broken chunks.

新的 mfs 文件修复工具集，该工具允许对部分丢失及损坏块的文件进行恢复

8.3.6 CGI scripts

First public version of CGI scripts allowing to monitor MFS installation from WWW browser.

第一个允许从 WWW 浏览器监控 MFS 安装的 CGI 脚本发布版本

8.4 MooseFS 官方 FAQ（TC 版）

What average write/read speeds can we expect?

The raw reading / writing speed obviously depends mainly on the performance of the used hard disk drives and the network capacity and its topology and varies from installation to installation. The better performance of hard drives used and better throughput of the net, the higher performance of the whole system.

In our in-house commodity servers (which additionally make lots of extra calculations) and simple gigabyte Ethernet network on a petabyte-class installation

on Linux (Debian) with goal=2 we have write speeds of about 20-30 MiB/s and reads of 30-50MiB/s. For smaller blocks the write speed decreases, but reading is not much affected.

Similar FreeBSD based network has got a bit better writes and worse reads, giving overall a slightly better performance.

Does the goal setting influence writing/reading speeds?

Generally speaking,

it doesn't. The goal setting can influence the reading speed only under certain conditions. For example, reading the same file at the same time by more than one client would be faster when the file has goal set to 2 and not goal=1.

But the situation in the real world when several computers read the same file at the same moment is very rare; therefore, the goal setting has rather little influence on the reading speeds.

Similarly, the writing speed is not much affected by the goal setting.

How well concurrent read operations are supported?

All read processes are parallel - there is no problem with concurrent reading of the same data by several clients at the same moment.

How much CPU/RAM resources are used?

In our environment (ca. 500 TiB, 25 million files, 2 million folders distributed on 26 million chunks on 70 machines) the usage of chunkserver CPU (by constant file transfer) is about 15-20% and chunkserver RAM usually consumes about 100MiB (independent of amount of data).

The master server consumes about 30% of CPU (ca. 1500 operations per second) and 8GiB RAM. CPU load depends on amount of operations and RAM on number of files and folders.

Is it possible to add/remove chunkservers and disks on fly?

You can add / remove chunkservers on the fly. But mind that it is not wise to disconnect a chunkserver if there exists a chunk with only one copy (marked in orange in the CGI monitor).

You can also disconnect (change) an individual hard drive. The scenario for this operation would be:

Mark the disk(s) for removal

Restart the chunkserver process

Wait for the replication (there should be no "undergoal" or "missing" chunks marked in yellow, orange or red in CGI monitor)

Stop the chunkserver process

Delete entry(ies) of the disconnected disk(s) in 'mfshdd.cfg'

Stop the chunkserver machine

Remove hard drive(s)

Start the machine

Start the chunkserver process

If you have hotswap disk(s) after step 5 you should follow these:

Unmount disk(s)

Remove hard drive(s)

Start the chunkserver process

If you follow the above steps work of client computers would be not interrupted and the whole operation would not be noticed by MooseFS users.

My experience with clustered filesystems is that metadata operations are quite slow. How did you resolve this problem?

We have noticed the problem with slow metadata operations and we decided to cache file system structure in RAM in the metadata server. This is why metadata server has increased memory requirements.

When doing df -h on a filesystem the results are different from what I would expect taking into account actual sizes of written files.

Every chunkserver sends its own disk usage increased by 256MB for each used partition/hdd, and a sum of these master sends to the client as total disk usage. If you have 3 chunkservers with 7 hdd each, your disk usage will be increased by 3*7*256MB (about 5GB). Of course it's not important in real life, when you have for example 150TB

of hdd space.

There is one other thing. If you use disks exclusively for MooseFS on chunkservers df will show correct disk usage, but if you have other data on your MooseFS disks df will count your own files too.

If you want to see usage of your MooseFS files use 'mfsdirinfo' command.

Do chunkservers and metadata server do their own checksumming?

Yes there is checksumming done by the system itself. We thought it would be CPU consuming but it is not really. Overhead is about 4B per a 64KiB block which is 4KiB per a 64MiB chunk (per goal).

What sort of sizing is required for the Master server?

The most important factor is RAM of mfsmaster machine, as the full file system structure is cached in RAM for speed. Besides RAM mfsmaster machine needs some space on HDD for main metadata file together with incremental logs.

The size of the metadata file is dependent on the number of files (not on their sizes). The size of incremental logs depends on the number of operations per hour, but length (in hours) of this incremental log is configurable.

1 million files takes approximately 300 MiB of RAM. Installation of 25 million files requires about 8GiB of RAM and 25GiB space on HDD.

When I delete files or directories the MooseFS size doesn't change. Why?

MooseFS is not erasing files immediately to let you revert the delete operation.

You can configure for how long files are kept in trash and empty the trash manually (to release the space). There are more details here:

http://moosefs.com/pages/userguides.html#2[MB1] in section "Operations specific for MooseFS".

In short - the time of storing a deleted file can be verified by the mfsgettrashtime

command and changed with mfssettrashtime.

When I added a third server as an extra chunkserver it looked like it started replicating data to the 3rd server even though the file goal was still set to 2.

Yes. Disk usage ballancer uses chunks independently, so one file could be redistributed across all of your chunkservers.

Is MooseFS 64bit compatible?Yes!

Can I modify the chunk size?

File data is divided into fragments (chunks) with a maximum of 64MiB each. The value of 64 MiB is hard coded into system so you cannot modify its size. We based the chunk size on real-world data and it was a very good compromise between number of chunks and speed of rebalancing / updating the filesystem. Of course if a file is smaller than 64 MiB it occupies less space.

Please note systems we take care of enjoy files of size well exceeding 100GB and there is no chunk size penalty noticeable.

How do I know if a file has been successfully written in MooseFS?

First off, let's briefly discuss the way the writing process is done in file systems and what programming consequences this bears. Basically, files are written through a buffer (write cache) in all contemporary file systems. As a result, execution of the "write" command itself only transfers the data to a buffer (cache), with no actual writing taking place. Hence, a confirmed execution of the "write" command does not mean that the data has been correctly written on a disc. It is only with the correct performance of the "fsync" (or "close" command that all data kept in buffers (cache) gets physically written. If an error occurs while such buffer-kept data is being written, it could return an incorrect status for the "fsync" (or even "close", not only "write" command.

The problem is that a vast majority of programmers do not test the "close" command status (which is generally a mistake, though a very common one). Consequently, a program writing data on a disc may "assume" that the data has been written correctly, while it has actually failed.

As far as MooseFS is concerned – first, its write buffers are larger than in classic file systems (an issue of efficiency); second, write errors may be more frequent than in case of a classic hard drive (the network nature of MooseFS provokes some additional

error-inducing situations). As a consequence, the amount of data processed during execution of the "close" command is often significant and if an error occurs while the data is being written, this will be returned in no other way than as an error in execution of the "close" command only.

Hence, before executing "close", it is recommended (especially when using MooseFS) to perform "fsync" after writing in a file and then check the status of "fsync" and – just in case – the status of "close" as well.

NOTE! When "stdio" is used, the "fflush" function only executes the "write" command, so correct execution of "fflush" is not enough grounds to be sure that all data has been written successfully – you should also check the status of "fclose".

One frequent situation in which the above problem may occur is redirecting a standard output of a program to a file in "shell". Bash (and many other programs) does not check the status of "close" execution and so the syntax of the "application > outcome.txt" type may wrap up successfully in "shell", while in fact there has been an error in writing the "outcome.txt" file. You are strongly advised to avoid using the above syntax. If necessary, you can create a simple program reading the standard input and writing everything to a chosen file (but with an appropriate check with the "fsync" command) and then use "application | mysaver outcome.txt", where "mysaver" is the name of your writing program instead of "application > outcome.txt".

Please note that the problem discussed above is in no way exceptional and does not stem directly from the characteristics of MooseFS itself. It may affect any system of files – only that network type systems are more prone to such difficulties. Technically speaking, the above recommendations should be followed at all times (also in case of classic file systems).

Janusz, [MB1]tu trzeba będzie zrobić prawidłowy link