Benchmarks & performance des infrastructures de stockage disque (au CC-IN2P3)

Loïc Tortay,

Réunion des sites LCG-France, CC-IN2P3 Lyon, 23 novembre 2010

Des benchmarks, pour quoi faire ?

- Appels d'offres
  - avant pour :
    - évaluer les besoins matériels
    - définir les critères de choix
  - pendant pour évaluer l'adéquation des propositions
  - après le choix pour la *recette*
- Veille technologique

Outils utilisés

- Historiquement : *micro-benchmarks* très simples presque toujours *a posteriori*
- Depuis 2004/2005, systématisation des critères de performances dans les appels d'offre
- Performance utile : vue de l'application, pas du système
- Nécessite de connaître le profil d'entrée/sorties des applications cibles
- Logiciels traditionnels généralement inadaptés (IOzone, Bonnie, Bonnie++, ...)
- Logiciels plus évolués disponibles mais trop lourds, complexes ou insuffisants (filebench, IOMeter, ...)
- Développements locaux : Xio, utilisé depuis 2005 pour tous les appels d'offres disques et les tests de matériel; mfiles, pour stresser les disques & machines (stabilité/fiabilité & consommation électrique)

Xio

- *Model-based* : définition précise des entrées/sorties réalisées pour pouvoir simuler les entrées-sorties d'une application
- *Multi-threadé*
- Simple et léger (+/-), multiples fonctionnalités :
  - Configuration explicite par *thread*, génération dynamique des configurations (scripts)
  - I/Os blocs (*read/write*), vecteurs de blocs (*writev/readv*) ou *mmap*
  - I/Os séquentielles, aléatoires, *stride* & *slide*
  - Possibilité d'utiliser des I/Os synchrones ou directes (*Direct-I/O*)
  - Temporisations & séquences d'entrées-sorties sur une cible
  - Modes d'exécution multiples, en particulier « temps fixé »
  - Entrées-sorties simultanées quelconques
  - Accès direct à des périphériques possible (sans système de fichiers)
  - Résultats CSV : une ligne (par type d'entrées/sorties) pour chaque thread
  - Statistiques détaillées continues (*time-series*) par thread optionnelles
  - ...

Xio v3

- Configuration plus simple : plus besoin de script de génération de configuration
- Options par thread/groupes de threads
- Séquences d'entrées/sorties sur plusieurs cibles (fichiers/devices)
- Trafic réseau en entrée et en sortie (réseau ↔ disque)
- OpenSource (licence Cecill-B, type ISC/BSD)
- Disponible sur git://git.in2p3.fr/xio.git (**pas encore**)

Profils d'entrées/sorties

- Caractériser les entrées/sorties des applications est non trivial, en particulier si les utilisateurs accèdent directement au stockage (accès transparent ou API +/- POSIX-like)
- Profils d'entrées-sorties utilisés pour les appels d'offres souvent très simples voire simplistes :
  - dCache : écritures/lectures séquentielles gros blocs (1 Mio), lectures/écritures simultanées (60%/40%)
  - Xrootd : écritures séquentielles gros blocs (*staging* depuis HPSS), lectures aléatoires petits blocs (16 Kio), lectures/écritures simultanées (95%/5%), variations avec temporisation ou *slide*
  - GPFS & HPSS : écritures/lectures séquentielles gros blocs (1 Mio), lectures/écritures simultanées (80%/20%), sur des *raw devices*
  - ...
- Critères de performances en **I/O par seconde par téraoctet utile**

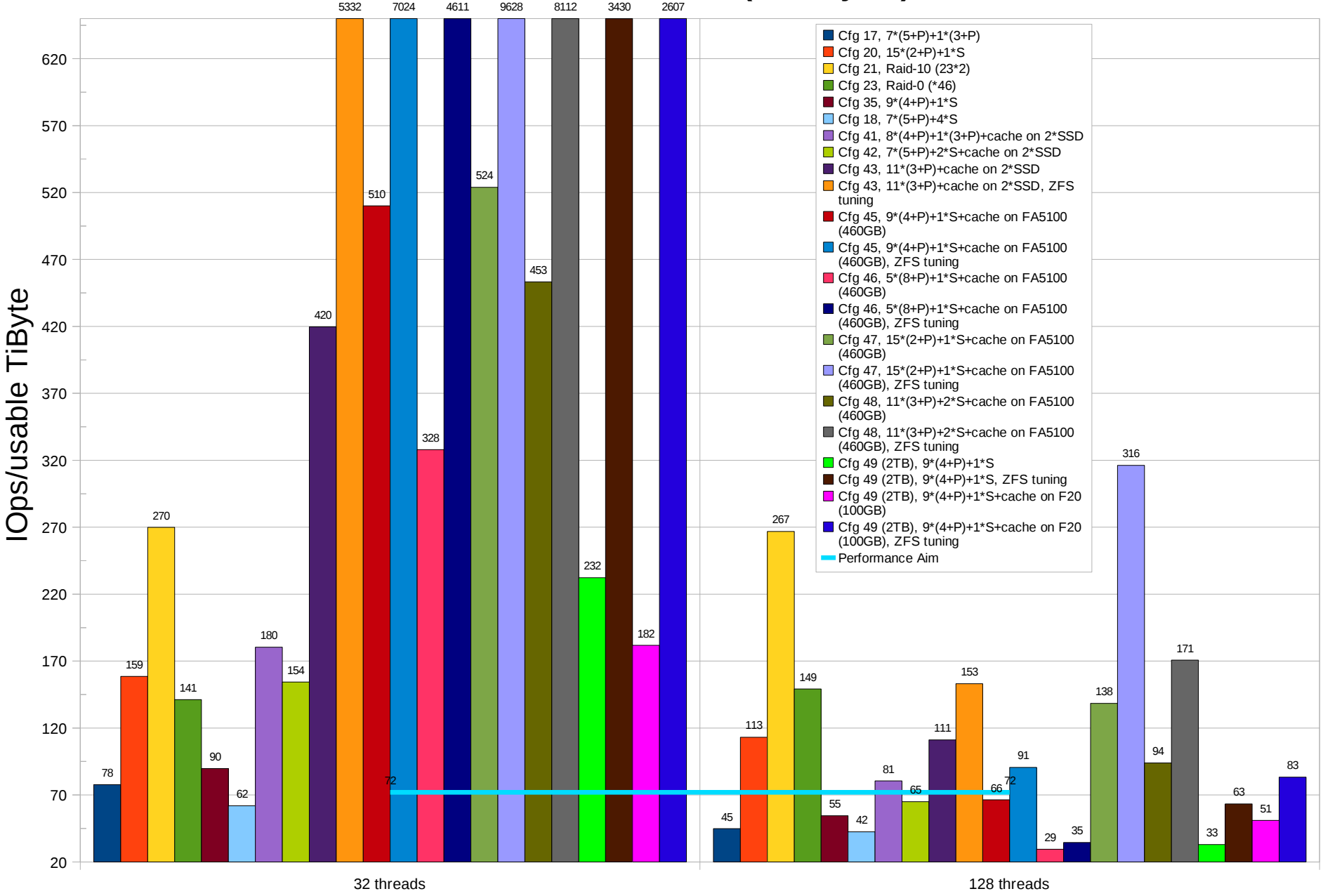
En pratique

- Une archive avec script(s) de lancement, sources de Xio, mini-documentation, scripts de synthèse des résultats
- Temps fixé : 20 à 30 minutes par passe, en général 3 passes : écriture, lecture, mixte; tests répétés avec un nombre croissant de threads
- Profil Xrootd utilisé pour la plupart des *gros* appels d'offre type *DAS* (Sun X4500, X4540, ...), 2009 : 72 IOps/Tio utile en lecture, 2010 : 48 IOps/Tio utile; 4 IOps/Tio utile en écriture (2009 & 2010)
- Profil dCache utilisé pour le dernier appel d'offre *DAS* : 9 IOps/Tio utile en lecture, 6 IOps/Tio utile en écriture
- Objectif principal : pouvoir envoyer/recevoir ce qui transite sur le réseau, base donnée par le débit des interface(s) réseaux
- Pas de sur-optimisation

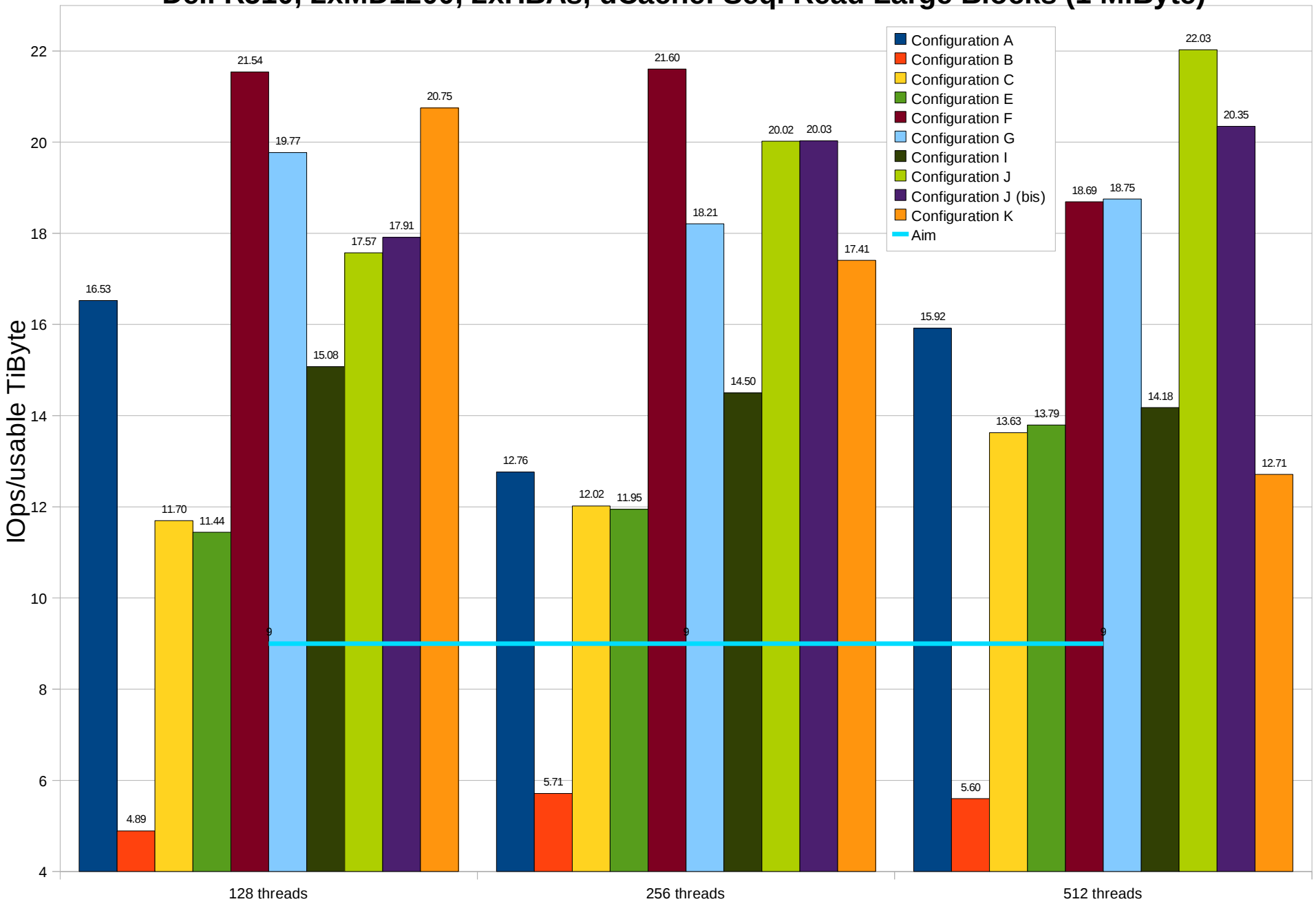
Critères d'évaluation

- Disque : ~150 IOps, 70 à 100 Mio/s profil favorable (séquentiel gros blocs), I/O minimale 512 octets (4 Kio)
- SSD/Flash : kIOps *random read* (ou plus Fusion-IO/Oracle F20/F5100) mais petite capacité ou coût très élevé, I/O minimale 4 Kio
- Ordres de grandeur RAID-[56] (heuristiques) :
  - IOps/disque * nombre de disques, si I/O > *chunksize* ou *stripesize*
  - IOps/disque, si I/O < *chunksize*
- RAID-5 acceptable si nombre de disques < 8
- RAID-6 si nombre de disques > 8 ou disques > 500 Go
- RAID-[56] & nombre de disques *vs Uncorrectable Bit Error Rate* ⇒ à partir de N disques suffisament capacitifs erreur « certaine », intérêt de *parity/checksum-check on read* (DDN, ZFS, Btrfs)
- Compromis performance/espace disponible/sécurité des données

X4540, Xrootd: Rand. Read Small Blocks (16 kiBytes)



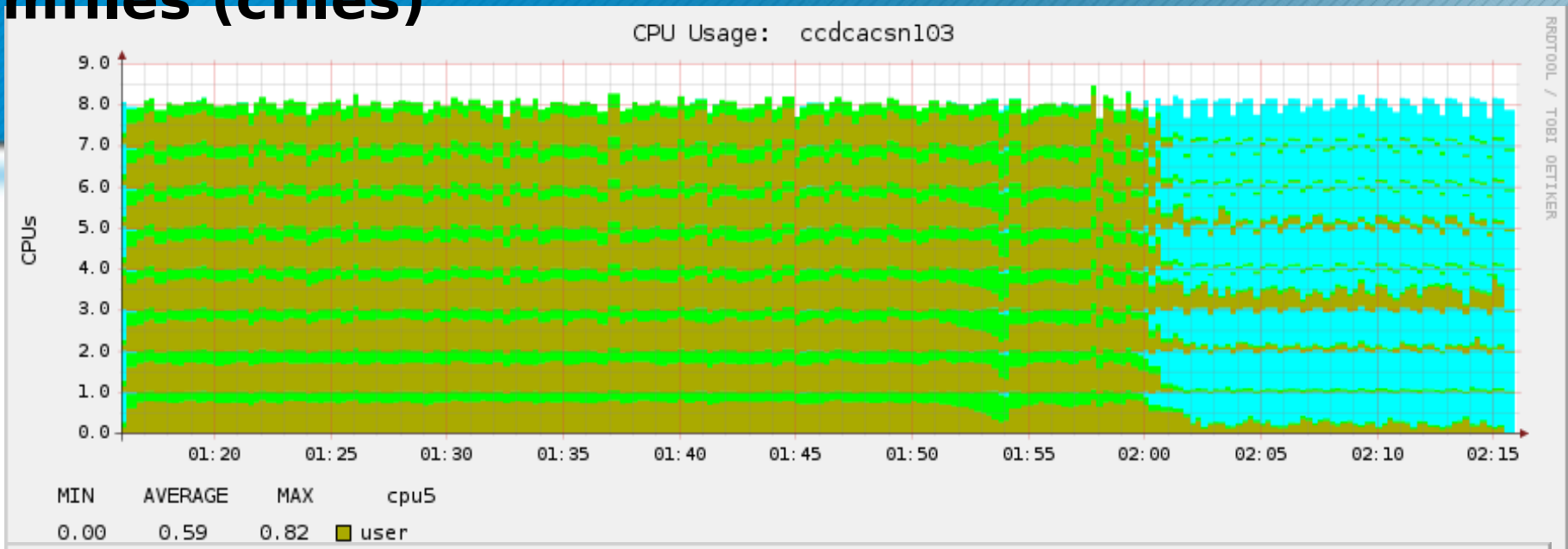
Dell R510, 2xMD1200, 2xHBAs, dCache: Seq. Read Large Blocks (1 MiByte)

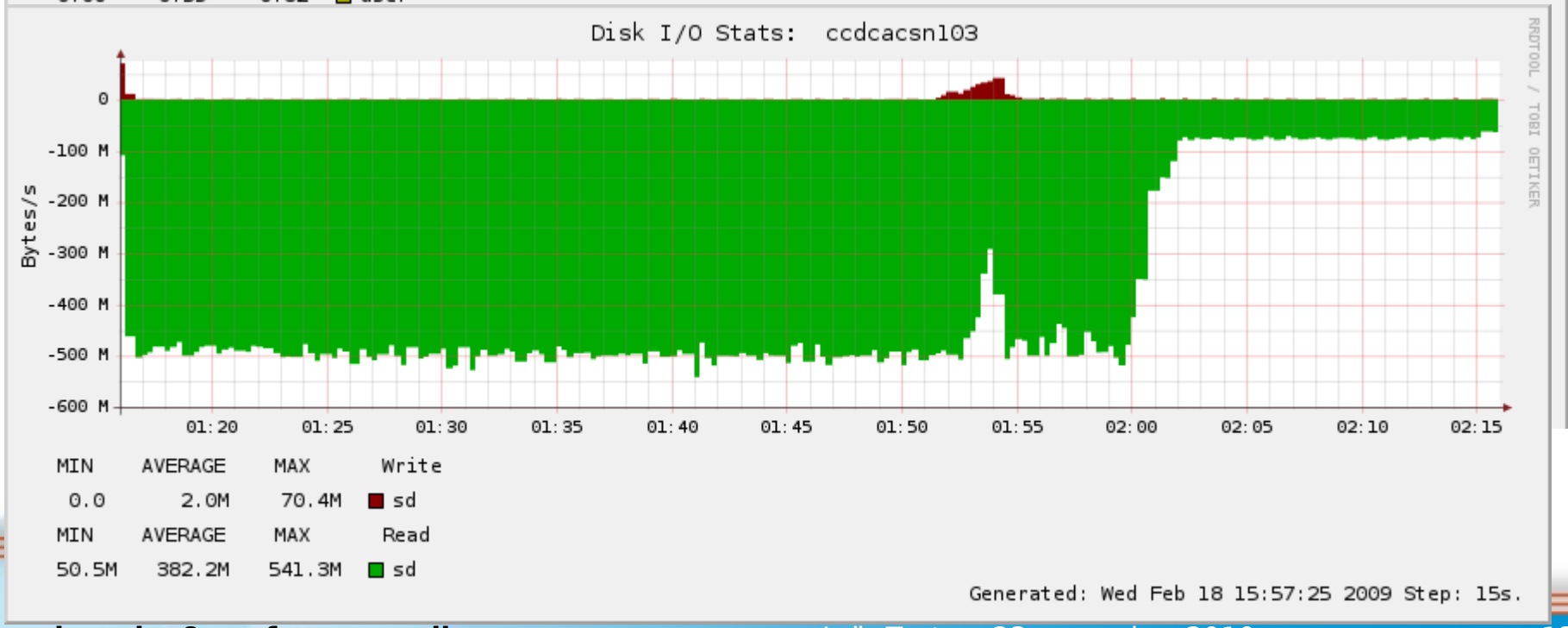


mfiles

- Programme de test/stress de systèmes de fichiers, utilisable aussi comme test de performance ou pour stress de machine et/ou disques
- Comment se comporte tel (type de) système de fichiers avec *n* millions de fichiers & des accès intensifs sur les méta-données ?
- Crée une hiérachie de répertoires/fichiers/liens symboliques selon un fichier de configuration, soit artificiel soit dérivé de l'état d'un système de fichier existant
- Multiples fonctionnalités :
  - *Multi-threadé*
  - Mode vérification : re-lecture de chaque fichier après écriture complète
  - Mode checksum/hash : calcul du *SHA-256* de chaque fichier et stockage dans un attribut étendu, vérification avec *cfiles*, consommation CPU élevée
  - En option : I/O synchrones ou directes, restauration des méta-données, contenu des fichiers aléatoire ou fixé, fichiers creux, noms d'objets & méta-données anonymisés, pseudo-fragmentation des fichiers, statistiques détaillées en continu par thread, ...

mfiles (cfiles)





Questions ?