開
放
的
열린
مفتوح
libre
मुक्त
ముక్త
livre
libero
ಮುಕ್ತ
开放的
açık
open
nyílt
⠠⠠⠠⠠⠠
פתוח
オープン
livre
ανοικτό
offen
otevřený
öppen
открытый
வெளிப்படை

open


Utiliser


Améliorer


Prêcher

License

Fonctionnalités de ZFS

- Modèle de stockage en pool
- Cohérence permanente sur le disque
- Protection contre la corruption des données
- Nettoyage des données en direct
- Aperçus et clones instantanés
- Sauvegarde et restauration natives rapides
- Grande évolutivité
- Compression intégrée
- Administration simplifié

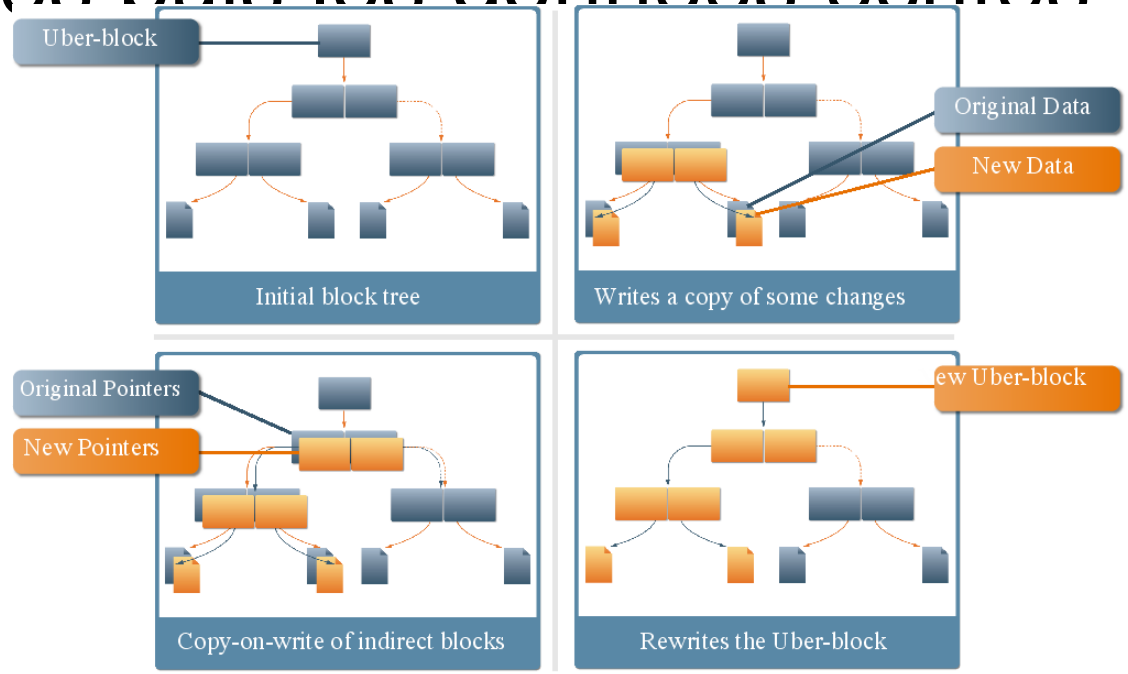
Modèle de stockage en pool

- Permet de s'émanciper de la couche Volume Manager traditionnelle, et offre des fonctionnalité de redondance

 # zpool create tank raidz c0t0d0 c0t1d0 c0t2d0 c0t3d0 c0t4d0 c0t5d0
Création d'un volume en Raid Z
 # zpool create tank mirror c0t0d0 c0t1d0 mirror c0t2d0 c0t3d0 spare c0t4d0
Création d'un volume composé d'un miroir de 2 disques avec un spare
 # zfs create tank/home
Création d'un file system
 # zfs set mountpoint=/export/home tank/home
Modification du point de montage, sera hérité
 # zfs create tank/home/bob
Création d'un nouveau fs
 # zpool add tank mirror c1t0d0 c1t1d0
Pour le faire grandir

Cohérence permanente sur le disque

- Schéma d'accès au disques transactionnel (COW), les données modifiées sont stockées sur d'autre blocs dans une première phase, ensuite les anciens blocks sont invalidés puis les données écrites avérées



Protection contre la corruption de données

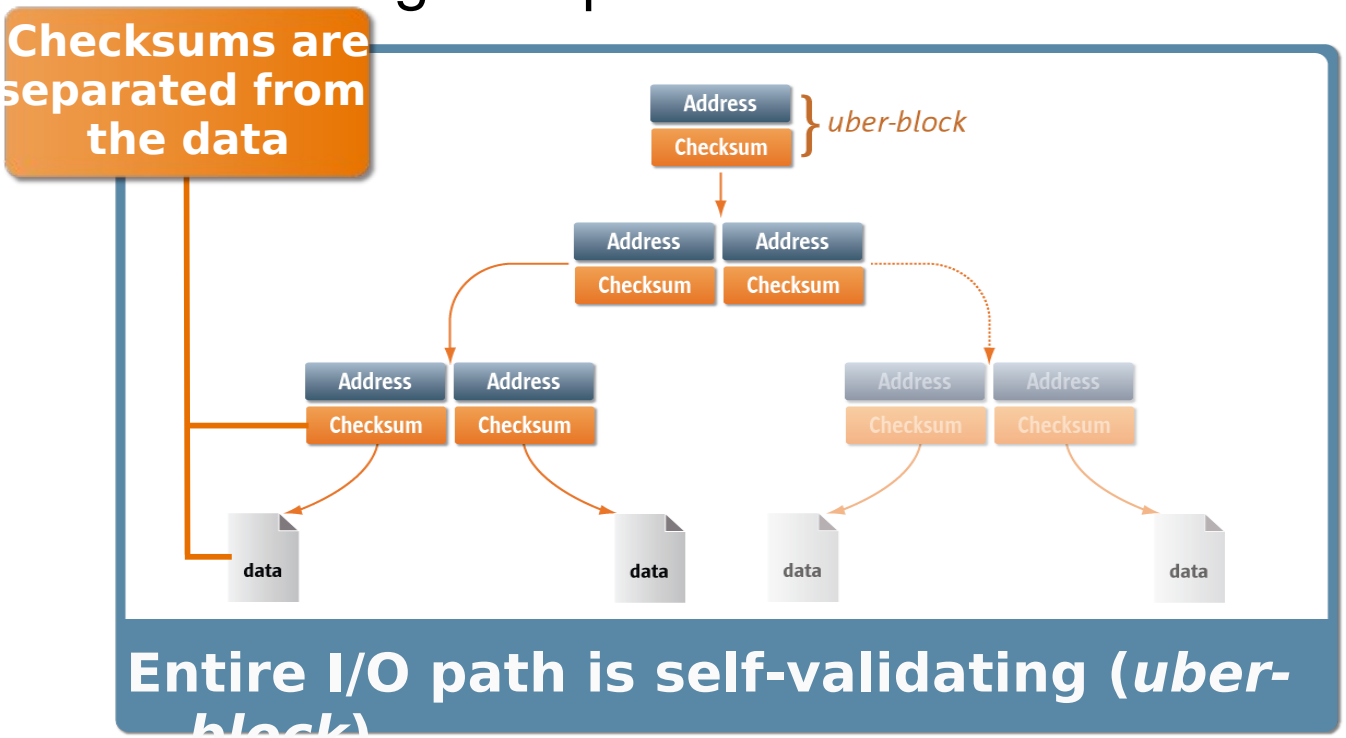
- s'appuie sur un calcul de parité (checksum sur 256 bits), au niveau block

  Lors des accès, ZFS vérifie la cohérence des blocks, et réparre automatiquement les erreurs rencontrées.

Nettoyage des données en direct

- recalcule des parités

  Un check global peut être lancé manuellement.



Prevents:

- Silent data corruption
- Panics from corrupted metadata
- Phantom writes
- Misdirected reads and writes
- DMA parity errors
- Errors from driver bugs
- Accidental

Aperçus et clones instantanés

- Possibilité de créer un nombre infini de snapshot, de clones
- Création d'un snap

  # zfs snapshot pool/home/bob@yesterday

  Le snap n'est pas visible automatiquement (configurable)
- Promotion d'un clone

  # zfs snapshot pool/project/production@today

  # zfs clone pool/project/production@today pool/project/beta

  Modifier les données dans /beta

  # zfs promote pool/project/beta

  # zfs rename pool/project/production pool/project/legacy

  # zfs rename pool/project/beta pool/project/production

  L'ancienne version n'est plus nécessaire

  # zfs destroy pool/project/legacy

Sauvegardes et restauration natives rapides

- 1 snapshot permet de réaliser 1 full backup, 2 snapshots permettent de réaliser un backup incremental => permet une réplication asynchrone

 # zfs send pool/fs@a | ssh host zfs receive poolB/received/fs@a
 # zfs send -i a pool/fs@b | ssh host zfs receive poolB/received/fs
Envoi d'un premier flux sur un autre serveur, puis d'un second incremental du premier

Grande évolutivité

- s'appuie sur un pipelining équivalent à ce qui est géré par les processeurs

  Plusieurs niveaux de cache peuvent être géré,par exemple des disques capacitifs et des disques SSD. Ces disques seront configurés spécialement, pour que ZFS puisse les utiliser pour un pool donné. Ensuite, lui va le gérer pour optimiser les IO par rapport au pool (priorité aux lectures, écriture en optimisant les caractéristiques des disques, ...). La conformité POSIX est ainsi assurée.

Compression intégrée

- activable et desactivable à tout moment, réalisée par fichier, configurée au niveau du file system

  # zfs set compression=off pool/home
  # zfs set compression=on pool/home/anne

- Compression peut-être égal à :
  - on
  - off
  - lzjb
  - Gzip
  - gzip-N

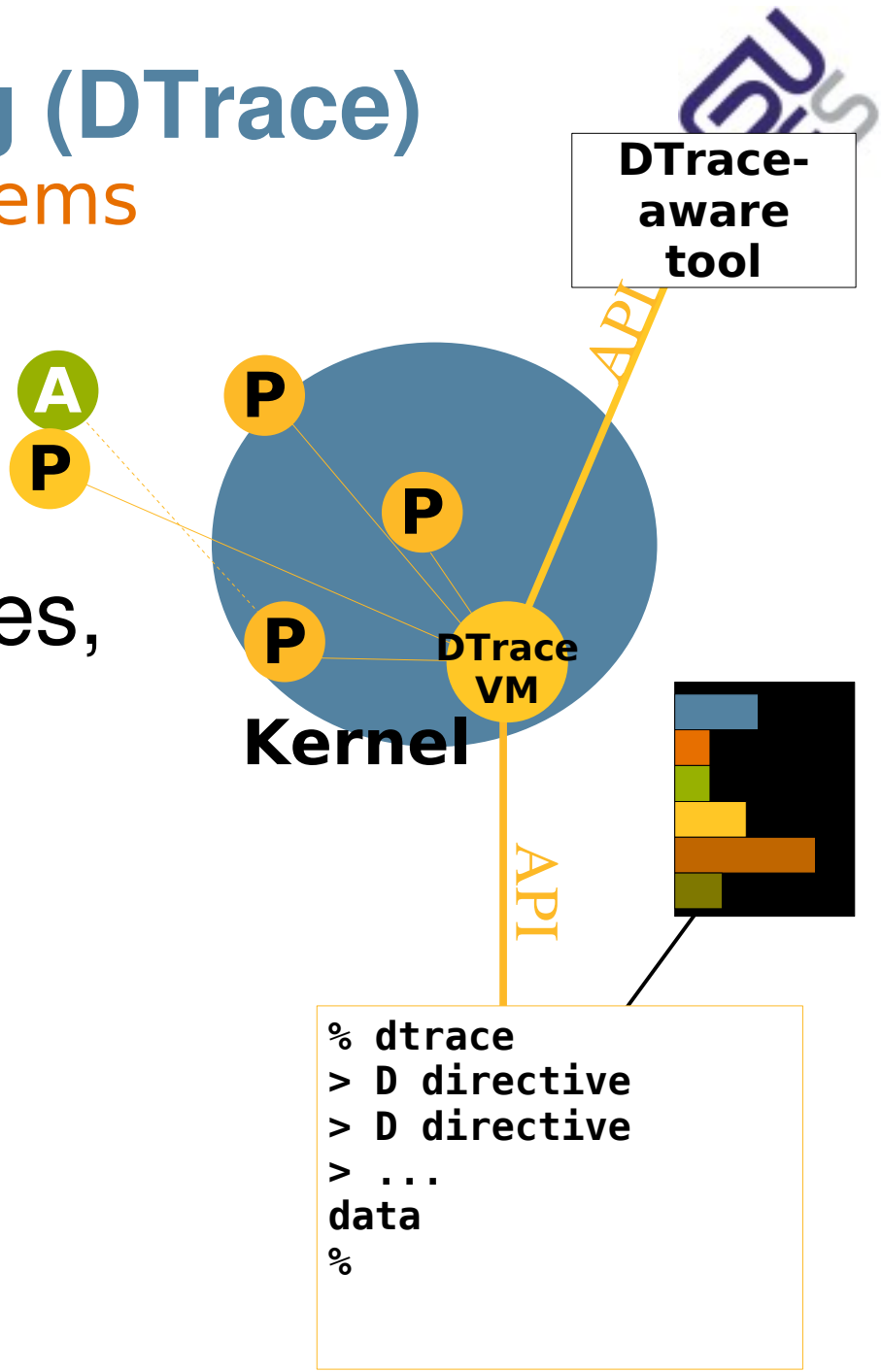
Le besoin

- You need tools to observe and debug different situations like: analysing the system performance, debugging an application or understanding the system utilisation or saturation, debugging a system/kernel crash
- A big number of observability and debug utilities under Solaris
- Several areas: Process Control, Process Statistics, Process Debugging, Kernel Debugging and Statistics and System

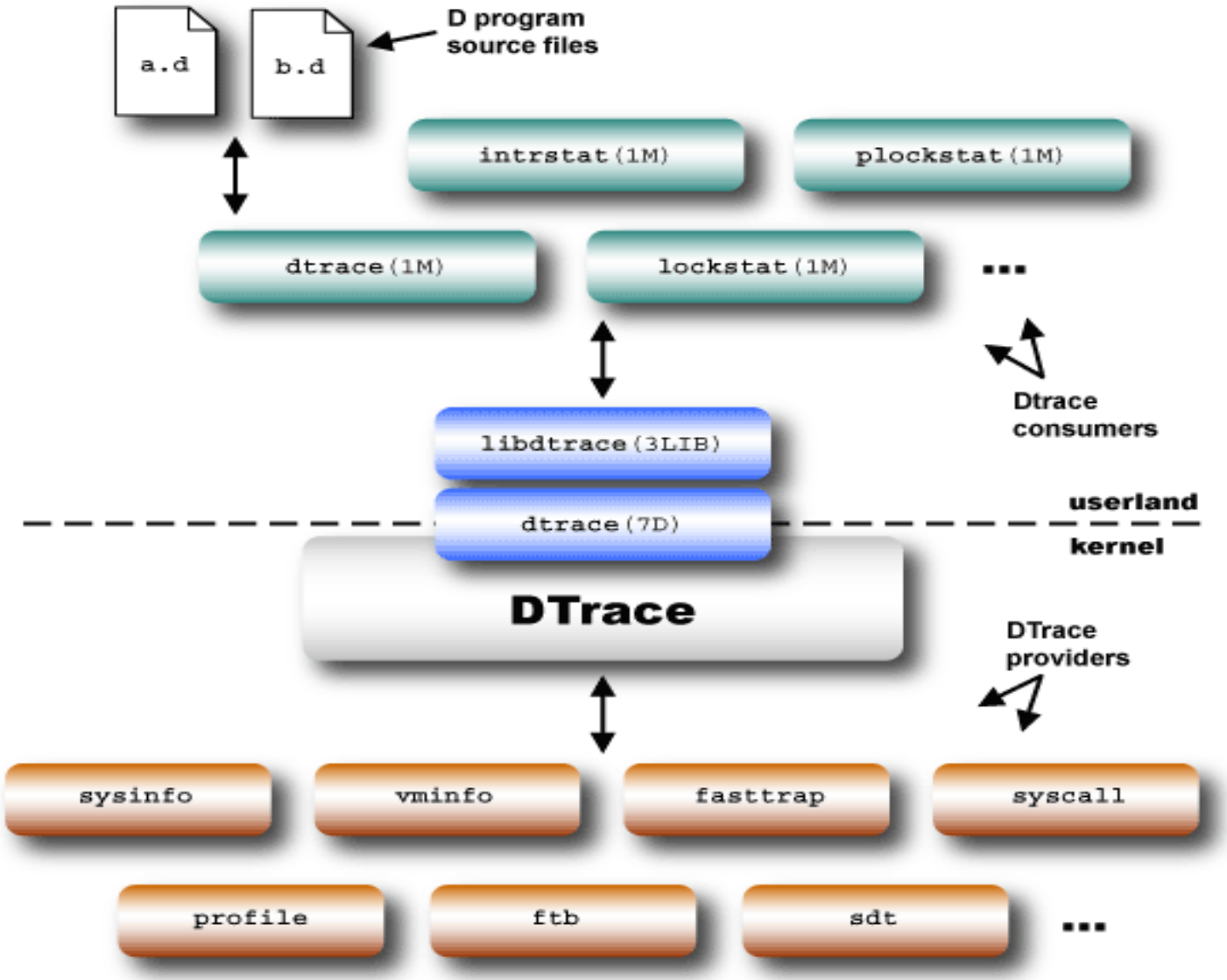
Solaris Dynamic Tracing (DTrace)

Designed for Production Systems

- Safe; always there
  - No performance hit
  - No app or OS changes
- Problems solved in minutes, not days
- Instrument every line in every application
- Views system as a whole
  - Comprehensive
  - Extensible; scriptable
- Massive performance



DTrace – Owerview



Quelques exemples

- System Calls Count by Application

```
$ dtrace -n 'syscall:::entry{@[execname] =
  count();}'
```

- System Calls Count by Application and Process

```
$ dtrace -n 'syscall:::entry{@[execname,pid]
  = count();}'
```

- How many times a file has been opened

```
$ dtrace -n
  'syscall::open:entry{@[copyinstr(arg0)] =
  count();}'
```

Quelques exemples

- Files Opened by process

```
$ dtrace -qn
  'syscall::open*:entry{ printf("%s
  %s\n",execname,copyinstr(arg0)); }'
```

- Read Bytes by process

```
$ dtrace -n 'sysinfo:::readch{ @[execname] =
  sum(arg0);}'
```

- Write Bytes by process

```
$ dtrace -n 'sysinfo:::writech{ @[execname]
  = sum(arg0);}'
```

Quelques exemples

- How big a read is

```
$ dtrace -n 'syscall::read:entry{@[execname]
  = quantize(arg2);}'
```

- How big a write is

```
$ dtrace -n
  'syscall::write:entry{@[execname] =
  quantize(arg2);}'
```

- Disk size by process

```
$ dtrace -qn 'io:::start{printf("%d %s
  %d\n",pid,execname,args[0]->b_bcount); }'
```

Quelques exemples

- High system time

```
$ dtrace -n profile-501'{@[stack()] =
  count()}END{trunc(@, 25)}'
```

- What processes are using fork

```
$ dtrace -n 'syscall::fork*:entry{printf("%s
  %d",execname,pid);}'
```

The toolkit: DTraceToolkit

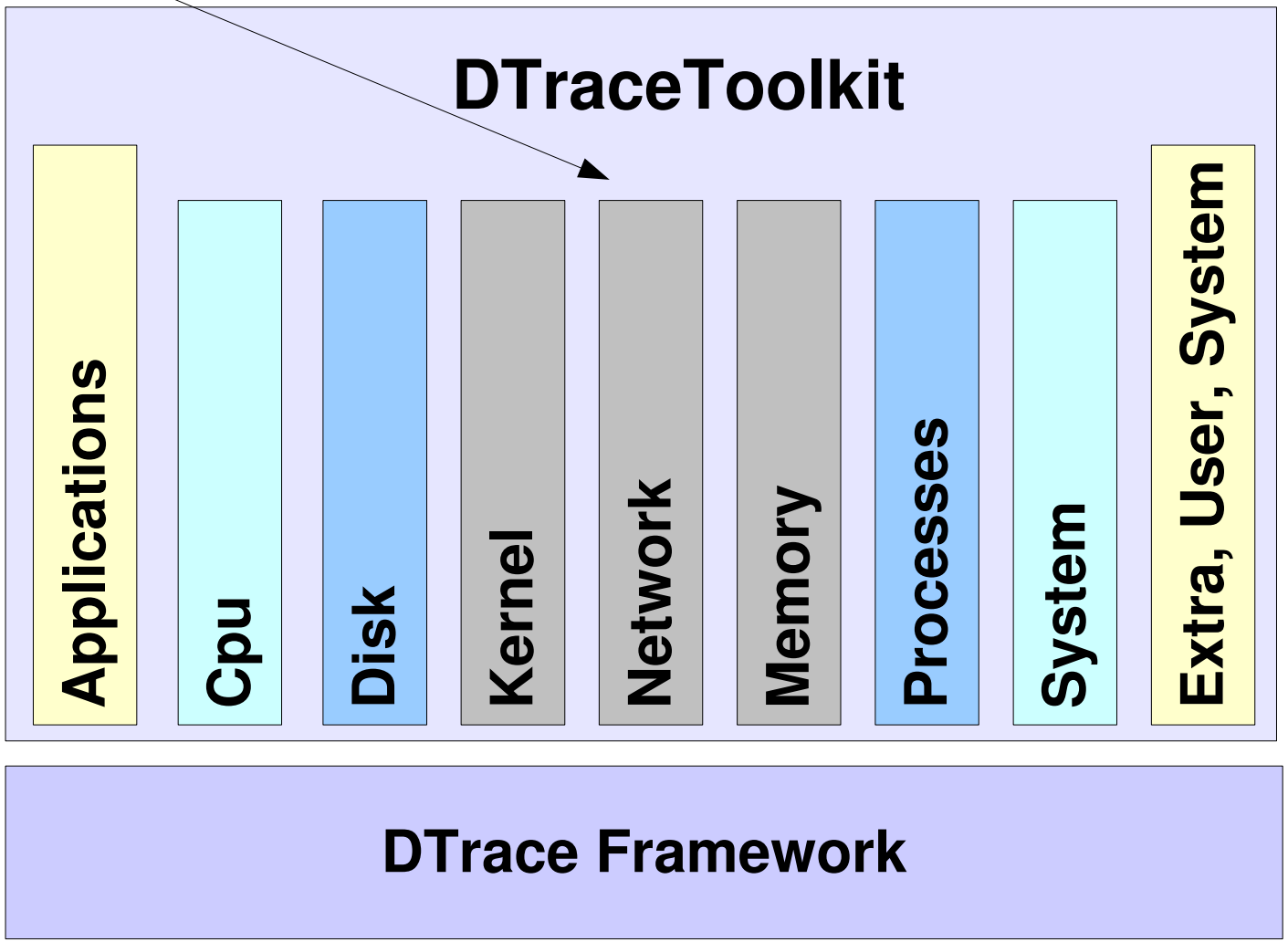
- Introduction
- Installation and Setup
- Toolkit elements
- Categories
- Free your mind
- Examples

Introduction

- The DTraceToolkit is a collection of useful documented scripts developed by the OpenSolaris DTrace community built on top of DTrace framework
- Available under www.opensolaris.org
- Ready DTrace scripts
- The toolkit contains:
  - the scripts
  - the man pages
  - the example documentation
  - the notes files

Introduction, cont.

Script Categories: collection of D scripts



The toolkit: DTraceToolkit

- Introduction
- **Installation and Setup**
- Toolkit elements
- Categories
- Free your mind
- Examples

Installation and Setup

- Download the toolkit

  http://www.opensolaris.org/os/community/dtrace/dtracetoolk

- Installation Notes
  - gunzip and "tar xvf" the file
  - run ./install – default installation /opt/DTT
  - read Guide to find out how to get started
  - a list of scripts is in Docs/Contents
- Setup DTT
  - PATH=$PATH:/opt/DTT/Bin
  - MANPATH=$MANPATH:/opt/DTT/Man

    (assuming the toolkit was installed in /opt/DTT)

The toolkit: DTraceToolkit

- Introduction
- Installation and Setup
- **Toolkit Elements**
- Categories
- Free your mind
- Examples

Toolkit Elements

```
DTraceToolkit-X.XX/
     Bin/                 Symlinks to the scripts
     Apps/                Application specific scripts
     Cpu/                 Scripts for CPU analysis
     Disk/                Scripts for disk I/O analysis
     Docs/                Documentation
        Contents          Command list for the Toolkit
        Examples/         Examples of command usage
        Faq               Frequently asked questions
        Links             Further DTrace links
        Notes/            Notes on Toolkit commands
        Readme            Readme for using the docs
     Extra/               Misc scripts
     Guide                This file!
     Kernel/              Scripts for kernel analysis
     License              The CDDL license
     Locks/               Scripts for lock analysis
     Man/                 Man pages
        man1m/            Man pages for the Toolkit commands
     Mem/                 Scripts for memory analysis
     Net/                 Scripts for network analysis
     Proc/                Scripts for process analysis
     System/              Scripts for system analysis
     User/                Scripts for user based activity analysis
     Zones/               Scripts for analysis by zone
     Version              DTraceToolkit version
     install              Install script, use for installs only
```

Toolkit Elements, cont.

- Categories
  - Apps – scripts for certain applications: Apache, NFS
  - Cpu – scripts for measuring CPU activity
  - Disk – scripts to analyse I/O activity
  - Extra – other categories
  - Kernel – scripts to monitor kernel activity
  - Locks – scripts to analyse locks
  - Mem – scripts to analyse memory and virtual memory
  - Net – scripts to analyse activity of the network interfaces, and the TCP/IP stack
  - Proc – scripts to analyse activity of a process
  - System – scripts to measure system wide activity
  - User – scripts to monitor activity by UID

Toolkit Elements, cont.

- Documentation
  - Man/: all scripts are documented as UNIX manual pages
  - Docs/: a generic place to find the documentation
  - Docs/Notes/: several short guides about toolkit's commands
  - Docs/Example/: examples of command usage
  - Docs/Content/: complete list of all commands
  - Docs/Faq/: DTT Frequently Asked Questions

The toolkit: DTraceToolkit

- Introduction
- Installation and Setup
- Toolkit Elements
- **Categories**
- Free your mind
- Examples

Categories

- **Applications**
  - Used to measure and report certain metrics from applications like: Apache Web server, NFS client, UNIX shell
  - **httpdstat.d**: computes real-time Apache web statistics: the number of connections, GET, POST, HEAD and TRACE requests
  - **nfswizard.d**: used to measure the NFS client activity regarding response time and file accesses
  - **shellsnoop**: captures keystrokes, used to debug and catch command output. Use with caution !
  - **weblatency.d**: counts connection speed delays, DNS lookups, proxy delays, and web server response time.

Categories, cont.

- **Cpu**
  - Reports and list the CPU activity like: cross calls, interrupt activity by device, time spent servicing interrupts, CPU saturation
  - **cputypes.d**: lists the information about CPUs: the number of physical install CPUs, clock
  - **loads.d**: prints the load average, similar to uptime
  - **intbycpu.d**: prints the number of interrupts by CPU
  - **intoncpu.d**: lists the interrupt activity by device; example: the time consumed by the ethernet driver, or the audio device
  - **inttimes.d**: reports the time spent servicing the

Categories, cont.

- **Cpu**
  - **xcallsbypid.d** – list the inter-processor cross-calls by process id. The inter-process cross calls is an indicator how much work a CPU sends to another CPU
  - **dispqlen.d** – dispatcher queue length by CPU, measures the CPU saturation
  - **cpuwalk.d** – identify if a process is running on multiple CPUs concurrently or not
  - **runocc.d** – prints the dispatcher run queue, a good way to measure CPU saturation

Categories, cont.

- **Disk**
  - Analyses I/O activity using the io provider from DTrace: disk I/O patterns, disk I/O activity by process, the seek size of an I/O operation
  - **iotop**: a top like utility which lists disk I/O events by processes
  - **iosnoop**: a disk I/O trace event application. The utility will report UID, PID, filename regarding for a I/O operation
  - **bitesize.d**: analyse disk I/O size by process
  - **seeksize.d**: analyses the disk I/O seek size by identifying what sort I/O operation the process is making: sequential or random

Categories, cont.

- **Disk**
  - **iofile.d**: prints the total I/O wait times. Used to debug applications which are waiting for a disk file or resource
  - **iopattern**: computes the percentage of events that were of a random or sequential nature. Used easily to identify the type of an I/O operation and the average, totals numbers
  - **iopending**: prints a plot for the number of pending disk I/O events. This utility tries to identify the "serialness" or "parallelness" of the disk behavior
  - **diskhits**: prints the load average, similar to uptime
  - **iofileb.d**: prints a summary of requested disk activity

Categories, cont.

- **FS**
  - Analyses the activity on the file system level: write cache miss, read file I/O statistics, system calls read/write
  - **vopstat**: traces the vnode activity
  - **rfsio.d**: provides statistics on the number of reads: the bytes read from file systems (logical reads) and the number of bytes read from physical disk
  - **fspaging.d**: used to examine the behavior of each I/O layer, from the syscall interface to what the disk is doing
  - **rfileio.d**: similar with rfsio.d but reports by file

Categories, cont.

- **Kernel**
  - Analyses kernel activity: DNLC statistics, CPU time consumed by kernel, the threads scheduling class and priority
  - **dnlcstat**: inspector of the Directory Name Lookup Cache (DNLC)
  - **cputimes**: print CPU time consumed by the kernel, processes or idle
  - **cpudist**: print CPU time distributions by kernel, processes or idle
  - **cswstat.d**: prints the context switch count and average
  - **modcalls.d**: an aggregation for kernel function calls

Categories, cont.

- **Kernel**
  - **dnlcps.d**: prints DNLC statistics by process
  - **dnlcsnoop.d**: snoops DNLC activity
  - **kstat_types.d**: traces kstat reads
  - **pridist.d**: outputs the process priority distribution. Plots which process is on the CPUs, and under what priority it is
  - **priclass.d**: outputs the priority distribution by scheduling class. Plots a distribution
  - **whatexec.d**: determines the types of files which are executed by inspected the first four bytes of the executed file

Categories, cont.

- **Locks**
  - Analyses lock activity using lockstat provider
  - **lockbydist.d**: lock distribution by process name
  - **lockbyproc.d**: lock time by process name

Categories, cont.

- **Memory**
  - This category analyses memory and virtual memory things: virtual memory statistics, page management, minor faults
  - **vmstat.d**: a vmstat like utility written in D
  - **vmstat-p.d**: a vmstat like utility written in D which does display what “vmstat -p” does: reporting the paging information
  - **xvmstat**: a much improved version of vmstat which does count the following numbers: free RAM, virtual memory free, major faults, minor faults, scan rate

Categories, cont.

- **Memory**
  - **swapinfo.d**: prints virtual memory info, listing all memory consumers related with virtual memory including the swap physical devices
  - **pgpginbypid.d**: prints information about pages paged in by process id
  - **minfbypid.d**: detects the biggest memory consumer using minor faults, an indication of memory consumption

Categories, cont.

- **Network**
  - These scripts analyse the activity of the network interfaces and the TCP/IP stack. Some scripts are using the mib provider. Used to monitor incoming
  - **icmpstat.d**: reports ICMP statistics per second, based on **mib**
  - **tcpstat.d**: prints TCP statistics every second, retrieved from the **mib** provider: TCP bytes received and sent, TCP bytes retransmitted
  - **udpstat.d**: prints UDP statistics every second, retrieved from the **mib** provider
  - **tcpsnoop.d**: analyses TCP network packets and prints the responsible PID and UID. Useful to detect

Categories, cont.

- **Network**
  - connections: prints the inbound TCP connections. This displays the PID and command name of the processes accepting connections
  - **tcptop**: display top TCP network packets by process. It can help identify which processes are causing TCP traffic
  - **tcpwdist.d**: measures the size of writes from applications to the TCP level. It can help identify which process is creating network traffic

Categories, cont.

- **Process**
  - Analyses process activity: system calls/process, bytes written or read by process, files opened by process,
  - **sampleproc**: inspect how much CPU the application is using
  - **threaded.d**: see how well a multithreaded application uses its threads
  - **writebytes.d**: how many bytes are written by process
  - **readbytes.d**: how many bytes are read by process
  - **kill.d**: a kill inspector. What how signals are send to what applications
  - **newproc.d**: snoop new processes as they are

Categories, cont.

- **Process**
  - **syscallbyproc.d** & **syscallbypid.d**: system calls by process or by PID
  - **filebyproc.d**: files opened by process
  - **fddist**: a file descriptor reporter, used to print distributions for read and write events by file descriptor, by process. Used to determine which file descriptor a process is doing the most I/O with
  - **pathopens.d**: prints a count of the number of times files have been successfully opened
  - **rwbypid.d**: reports the no. of read/writes calls by PID
  - **rwbytype.d**: identifies the vnode type of read/write

Categories, cont.

- **Process**
  - **sigdist.d**: prints the number of signals received by process and the signal number
  - **topsysproc**: a report utility listing top number of system calls by process
  - **pfilestat**: prints I/O statistics for each file descriptor within a process. Very useful for debug certain processes
  - **stacksize.d**: measures the stack size for running threads
  - **crash.d**: reports about crashed applications. Useful to identify the last seconds of a crashed application

Categories, cont.

- **System**
  - Used to measure system wide activity
  - **uname-a.d**: simulates 'uname -a' in D
  - **syscallbysysc.d**: reports a total on the number od system calls on the system
  - **sar-c.d**: reports system calls usage similar to 'sar -c'
  - **topsyscall**: prints a report of the top system calls on the system

開
放
的
열린
مفتوح
libre
मुक्त
ముక్త
livre
libero
ಮುಕ್ತ
开放的
açık
open
nyílt
⠠⠠⠠⠠⠠
פתוח
オープン
livre
ανοικτό
offen
otevřený
öppen
открытый
வெளிப்படை

open


Utiliser


Améliorer


Prêcher