第1章

常用工具简介

性能调优是一个涉及面很广的话题，影响性能的因素很多，如网络、CPU、内存、I/O、应用程序等，这些因素都可能影响系统的整体性能。所以对性能的调优也并不是单纯地从一方面入手就能解决的问题，这如同木桶原理，系统能够运行多快，取决于最短的那一块板，而最短的那一块板就是经常所说的“瓶颈”。一般解决问题的思路是首先发现“瓶颈”，然后想办法解决。在发现的过程中，对系统的监控是十分必要的。本章将介绍一些常用的监控工具和 Spark 开发过程中辅助的框架。本章主要内容如下：

- Linux 中的性能监控命令。
- Prometheus。
- Grafana。
- Alluxio 的使用。

1.1 Linux 中的性能监控命令

在生产环境中，大部分应用程序都部署在 Linux 系统当中。在对 Spark 性能监控的过程中，对 Linux 系统本身的各种运行状态的监控也是必不可少的。本节基于 Linux 发行版 CentOS 7 介绍 Linux 中常用的性能监控命令和工具。

1.1.1 程序准备

为了更清楚地展示 Java 应用程序对系统 CPU、内存、I/O 等各方面的使用情况，突出各种监控命令对系统不同方面的监控情况，本节使用 4 个不同的 Java 程序，分别模拟真实业务对系统 CPU、内存、磁盘 I/O、网络 I/O 等方面的占用情况。

1. CPU 占用

```
public class CPUConsume {
```

```
    public static void main(String[] args) {
        int threadNumber = 1;
        if (args.length > 0) {
            threadNumber = Integer.parseInt(args[0]);
        }
        IntStream.range(0, threadNumber).forEach(i -> {
            new Thread(() -> {
                while (true) {
                }
            }).start();
        });
    }
}
```

此程序默认开启一个线程，使 CPU 陷入无限忙循环，大量占用 CPU 资源；同时可接收一个整型参数，开启指定的线程数，更大程度地占用 CPU 资源。

2. 内存占用

```
public class MemoryConsume {
    public static void main(String[] args) {
        List<byte[]> list = new ArrayList<>();
        int mbSize = 1024 * 1024;
        Runtime runtime = Runtime.getRuntime();
        while (true) {
            list.add(new byte[1024 * 1024]);
            if (list.size() % 10 == 0) {
                String format = "maxMemory: % sM totalMemory: % sM freeMemory: % sM";
                System.out.println(String.format(format, runtime.maxMemory()/mbSize,
runtime.totalMemory()/mbSize, runtime.freeMemory()/mbSize));
            }
        }
    }
}
```

此程序模拟消耗系统内存，其每次创建 1MB 数组，放入集合中。并且已分配的内存不可进行垃圾回收，直到系统 JVM 无法再分配更多的内存，将内存消耗殆尽。

3. 磁盘 I/O 占用

```
public class DiskIOConsume {
    public static void main(String[] args) {
        int threadNumber = 1;
        if (args.length > 0) {
            threadNumber = Integer.parseInt(args[0]);
        }
        byte[] data = new byte[1024 * 1024];
        IntStream.range(0, threadNumber).forEach(i -> {
            new Thread(() -> {
                while (true){
                    OutputStream out = null;
                    try {
                        out = new FileOutputStream(new File("/tmp/" + i));
                        out.write(data);
                    } catch (Exception e) { e.printStackTrace(); } finally {
```

```
                        if(out!= null){
                            try { out.close(); } catch (IOException e) { e.printStackTrace(); }
                        }
                    }
                }

            }).start();
        });
    }
}
```

此程序默认开启一个线程，循环向磁盘的/tmp 目录写入 1MB 数组；同时可以接收整型参数，开启更多的线程同时向磁盘写入数据。

4. 网络 I/O 占用

网络 I/O 的程序分为服务端程序和客户端程序。程序启动后，客户端将向服务端写入大量数据。

- 服务端程序。

```
public class NetConsumeServer {
    public static void main(String[] args) throws Exception {
        ServerSocket serverSocket = new ServerSocket(10010);
        while (true) {
            Socket socket = serverSocket.accept();
            new Thread(() -> {
                try {
                    byte[] buffer = new byte[2048];
                    InputStream inputStream = socket.getInputStream();
                    int i = 0;
                    while ((i = inputStream.read(buffer)) !=- 1) { }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }).start();
        }
    }
}
```

- 客户端程序。

```
public class NetConsumeClient {
    public static void main(String[] args) {
        int threadNumber = 1;
        if (args.length > 1) {
            threadNumber = Integer.parseInt(args[1]);
        }
        IntStream.range(0, threadNumber).forEach((i) -> {
            try {
                Socket socket = new Socket(args[0], 10010);
                OutputStream outputStream = socket.getOutputStream();
                while (true) {
                    outputStream.write(new byte[1024]);
                }
```

```
                } catch (IOException e) {
                    e.printStackTrace();
                }
            });
        }
    }
```

注意：本节中的程序将会严重消耗系统资源，可能会对系统造成不可预知的后果，请在测试环境中执行。

1.1.2 top 命令

top 命令是 Linux 系统中自带的一个应用程序，用于提供对系统运行状态的实时监控，可以显示系统的概要信息、各个进程或线程运行状态和资源占用情况。top 命令还内置了一些交互式命令，用于调整数据输出的方式，如按资源占用排序等。

1. 命令选项

top 命令的格式如下：

```
top [选项]
```

top 命令中的选项很多，表 1.1 列出一些常用的选项进行说明。

表 1.1 top 命令常用选项说明

选　项	说　明	选　项	说　明
-p	监控指定的进程	-d	指定两次数据刷新的间隔
-i	不显示空闲或僵尸进程	-u	按用户过滤进程
-c	显示启动进程命令的完整路径	-H	显示线程而不是进程
-o	按照某个字段排序显示		

2. 交互式命令说明

在 top 命令运行的过程中，还可以输入一些简单的交互式命令，以改变程序的输出状态。这些命令都是由一个字母或数字组成，常用的交互式命令如表 1.2 所示。

表 1.2 top 常用的交互式命令

命 令 名 称	说　明
h 或?	显示帮助信息
M	按照任务使用内存进行排序输出
P	按照任务 CPU 使用率排序输出
H	显示进程和线程的切换
s	改变 top 刷新频率，单位是秒。可以输入小数，如 0.5
c	显示进程启动命令完整路径
F/O	指定某个字段进行排序
1	显示所有 CPU 的运行状态
W	将当前设置写入~/.toprc 文件

使用交互式命令对输出形式调整好后，可以使用 W 命令进行保存。下次启动 top 程序时可自动加载此配置。如果想恢复默认配置，只需将配置文件～/.toprc 删除即可。

3. 示例

【例 1.1】 在命令行中输入不带任何参数的 top 命令即可启动 top 程序。

```
top - 14:59:02 up  5:47,  3 users,  load average: 0.01, 0.03, 0.05
Tasks: 297 total,   1 running, 296 sleeping,        0 stopped,   0 zombie
%CPU(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,         0.0 wa,   0.0 hi,   0.0 si,   0.0 st
KiB Mem :  26356841 + total,   26121228 + free,   2104036 used,   252104 buff/cache
KiB Swap:    4194300 total,     4194300 free,         0 used. 26064505 + avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME + COMMAND
 1825 ceph      20   0 1136408 436728  19100 S   0.3  0.2  19:34.29 ceph - osd
    1 root      20   0  192488   5460   2456 S   0.0  0.0   0:07.85 systemd
    5 root       0 - 20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
```

top 命令的输出中共包含两部分，由空行隔开。中间的空行为交互式命令的命令提示栏，可输入交互式命令。

上半部分为系统的概要信息，共有 5 行，每行的具体含义如下。

第一行显示系统启动的时间，当前登录的用户数，在过去 1min、5min、15min 内系统的负载情况。

第二行显示进程或线程的运行情况，依次为所有进程数量、运行进程数量、休眠进程数量、停止进程数量和僵尸进程数量。这一行显示为进程还是线程取决于当前的运行模式，当为进程模式时，显示当前正在运行的线程数；当运行为线程模式时，Tasks 将变为 Threads。使用交互式命令 H 可以切换进程和线程模式。

第三行显示 CPU 的运行状态。us 表示用户空间程序占用 CPU 的百分比，sy 表示内核空间占用 CPU 的百分比，ni 表示改变过优先级的进程占用 CPU 的百分比，id 表示 CPU 空闲的百分比，wa 表示 I/O 等待占用 CPU 的百分比，hi 表示硬件中断占用 CPU 的百分比，si 表示软件中断占用 CPU 的百分比，st 表示虚拟化占用前虚拟机的时间。

第四行显示内存的使用情况，分别为总内存、空闲内存、已使用内存、缓冲和缓存占用的内存。这部分显示默认以 KB 为单位，可通过交互式命令 E 切换显示的单位。

第五行显示交换分区使用的情况，分别为总交换分区内存、空闲交换分区内存、已使用交换分区的内存。最后一个为系统总的可用内存，会在空闲内存的基础上加上缓冲和缓存占用的内存等可以回收的内存。

下半部分显示系统中各个进程的详细情况。在显示的列表中，每一列的含义如下。

- PID：进程的 ID。
- USER：运行程序的用户。
- PR：进程的优先级，内核空间使用。
- NI：进程的 nice 值，用户空间使用。nice 值会影响进程的 PR 优先级。nice 值的范围为－20～19，数值越小，优先级越高。程序运行过程中可对进程的 nice 值进行调节。
- VIRT：进程使用的总的虚拟内存。

- RES：进程实际占用的物理内存。
- SHR：进程占用的共享内存。
- S：进程的运行状况，R 表示正在运行，S 表示休眠，Z 表示僵死状态。
- %CPU：进程 CPU 使用率。
- %MEM：进程占用的物理内存和总内存的百分比。
- TIME+：该进程启动后占用 CPU 的总时间。
- COMMAND：启动进程的命令。

【例 1.2】 在系统中运行 CPU 占用程序，使用 top 命令查看系统运行状态。

使用 java CPUConsume 6 启动程序，开启 6 个线程同时进入无限循环。启动 top 后，使用 P 交互指令按 CPU 使用率降序排序，使用 c 指令显示程序启动的完整命令。使用 top 命令查看输出如下：

```
Tasks: 224 total,   1 running, 223 sleeping,   0 stopped,   0 zombie
%CPU(s): 25.0 us,  0.0 sy,  0.0 ni,  75.0 id,   0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 13186345+total, 13057746+free,   854732 used,   431264 buff/cache
KiB Swap:          0 total,          0 free,          0 used. 13023400+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4070 root      20   0 35.269g  45432  11304 S 599.3  0.0   1:39.30 java
 4023 root      20   0  157820   2332   1552 R   0.3  0.0   0:00.07 top
    1 root      20   0  192324   5360   2508 S   0.0  0.0   0:01.90 systemd
```

由以上输出的第 3 行可知，当前系统 CPU 用户空间占用 25%，空闲 75%。运行的 Java 进程 PID 为 4070，其虚拟内存为 35.269GB，实际占用物理内存为 45 432KB，使用共享内存为 11 304KB，占用 CPU 为 599.3%，因为程序内部启动了 6 个线程，因此 CPU 的 6 个核心可以同时并行处理这 6 个线程，进而将 6 个核心跑满。

【例 1.3】 在系统中运行磁盘 I/O 占用程序，使用 top 命令查看系统运行状态。

使用 java DiskIOConsume 100 启动程序，该程序开启了 100 个线程同时向磁盘写入数据。使用 top 命令查看输出如下：

```
top - 10:38:15 up 12:33,  2 users,  load average: 51.92, 15.71, 5.73
Tasks: 224 total,   1 running, 223 sleeping,   0 stopped,   0 zombie
%CPU(s):  0.5 us,  2.4 sy,  0.0 ni,  0.7 id,   95.4 wa,  0.0 hi,  0.0 si,  0.9 st
KiB Mem : 13186345+total, 13019052+free,  1149112 used,   523824 buff/cache
KiB Swap:          0 total,          0 free,          0 used. 12993879+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4115 root      20   0 41.236g 328748  11832 S  91.7  0.2   0:37.04 java
  707 root      20   0  214248   3640   2920 S   0.3  0.0   0:01.91 rsyslogd
 4253 root      20   0  157820   2328   1548 R   0.3  0.0   0:00.06 top
    1 root      20   0  192324   5360   2508 S   0.0  0.0   0:01.90 systemd
```

由以上输出的第三行可知，当前系统用户空间 CPU 占用 0.5%，内核空间 CPU 占用 2.4%，wa 中参数显示 I/O 等待的 CPU 占比达到 95.4%。一般 I/O 等待 CPU 占比过高说明当前系统 I/O 特别频繁，当前状态下磁盘 I/O 压力过大。

1.1.3　htop 命令

htop 命令与 top 命令功能类似，是一个交互式的进程查看工具。htop 与 top 相比其界面显示更加友好，操作更加人性化。htop 与 top 的不同之处主要有以下几点。

- htop 界面显示更加直观，支持个性定制化。
- htop 支持鼠标操作。
- htop 支持彩色显示、主题选择等。
- htop 有滚动列表，可滚动查看完整内容。
- htop 直接通过界面结束某个进程。

1. 软件安装

htop 并不是 Linux 系统中自带的命令，使用 htop 需要进行安装。在 EPEL 源中包含 htop 的安装包及依赖包信息，可通过 EPEL 源安装 htop。

安装 EPEL 源命令如下：

```
yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm -y
```

安装 htop 命令如下：

```
yum install htop -y
```

安装完成后，输入 htop 命令，若显示 htop 主界面，则为安装成功。htop 的主界面如图 1.1 所示。

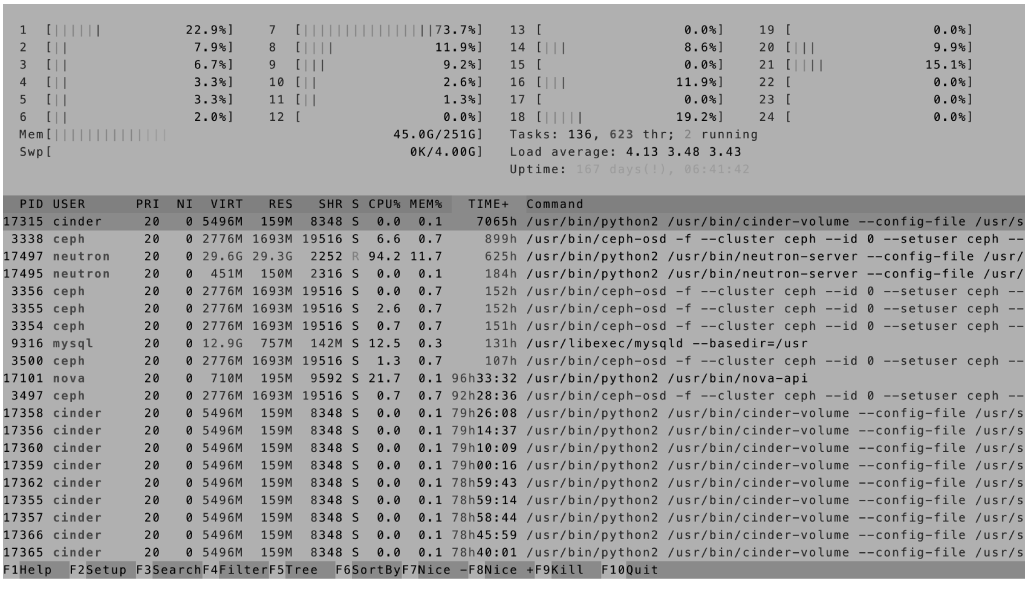

图 1.1　htop 的主界面

2. 命令选项

htop 命令的格式如下：

```
htop [选项]
```

htop 命令中的选项不太常用，一般直接输入 htop 命令即可。其各种选项功能在进入 htop 命令后，使用交互式操作都可以实现。htop 命令的常用选项如表 1.3 所示。

表 1.3 htop 命令的常用选项

选　项	说　明	选　项	说　明
-C	以黑白色彩显示	-t	指定两次数据刷新的间隔
-d	设置两次刷新时间间隔，单位为 0.1s	-u	按用户过滤进程
-h	显示启动进程命令的完整路径	-p	显示指定的进程
-s	按照某个字段排序显示	-v	显示 htop 版本号

3. 主界面介绍

htop 的主界面包括 3 部分，如图 1.1 所示。

其中顶部又分为左、右两部分，左边显示 CPU 的使用情况、内存的使用率、交换分区的使用率，右边显示 CPU 的使用情况、运行的任务的总进程数、线程数以及正在运行的任务数。细心的读者可能会发现 htop 显示的进程数与 top 显示的进程数是不同的，这是因为 htop 默认隐藏了系统内核的进程，使得 htop 显示的进程数较少。

中间部分显示的是系统中各个进程的详细情况，这部分和 top 命令显示的基本相同。但是在 htop 中，这一部分中每一列的名称是可以用鼠标单击的，以实现对某一列数值的排序显示；也可以单击某一行，从而选中某一个进程，实现对该进程进一步操作。

底部显示的是交互式命令的快捷键，分别为 F1～F10 及对应的功能。

4. 交互式命令

在主界面底部的交互式命令中，其功能都是自解释的。这些功能除了可以按相应的按键完成，还可以通过单击鼠标完成。

F1 可查看帮助信息。在帮助信息中可知，除了自带的交互式命令，常用的命令还有 M、P，可按照内存和 CPU 使用率进行排序。

F2 键可对主界面显示的内容进行设置，其界面如图 1.2 所示。如在 Meters 选项中，可设置在主界面的顶部需要显示的内容，还可以分别设置左边和右边显示的内容。在 Display options 选项中，可以设置主界面显示的方式，如是否显示系统进程、是否以进程树模式显示、是否显示程序的路径等。在 Colors 选项中，可以设置系统的主题颜色。在 Columns 选项中，可以设置进程列表中每一列具体显示的内容，从而修改系统默认显示的内容。

F3 键可实现对某个进程按照名称进行搜索，使搜索出的结果处于选中状态，方便对该进程进行再次操作。

F4 键为过滤功能，可通过该功能按照名称过滤出需要查看的进程。按 Esc 键可取消过滤操作。

F5 键可实现进程树模式和排序模式的切换。在进程树模式中，各个进程之间的关系可

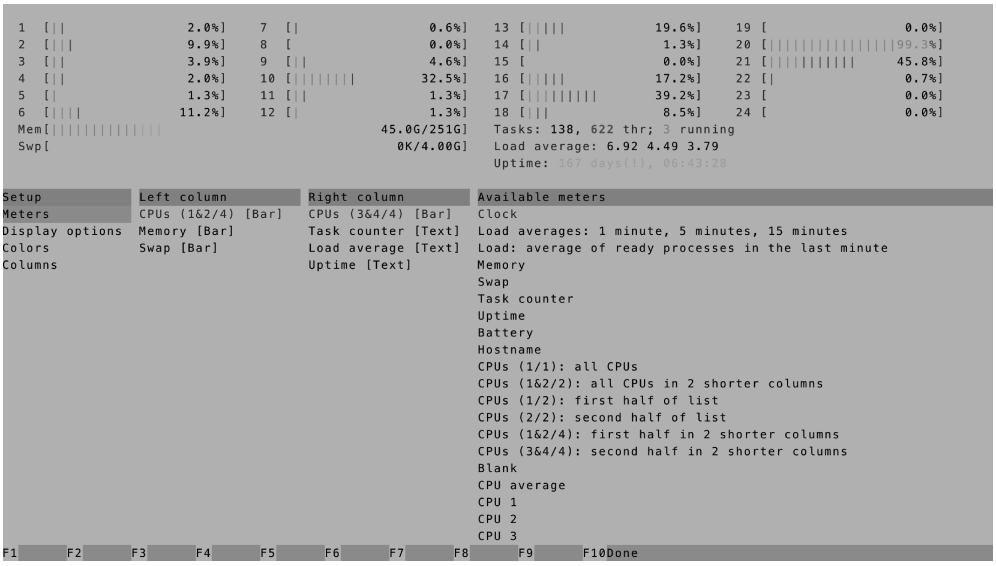

图 1.2　htop 设置界面

通过树状结构显示。在排序模式中可按 F6 键选择排序的字段，将进程列表按照指定的字段排序后进行显示。此外还可以直接通过鼠标单击对应的字段名称实现排序。

F7/F8 键可以加/减进程对应的 nice 值，通过修改进程的 nice 值，进而改变进程的优先级。

F9 键可直接终止当前选中的进程，而不用像 kill 一样需要指定对应的进程 id。

1.1.4　vmstat 命令

vmstat 也是 Linux 自带的一个性能监控工具，vmstat 为 virtual memory statistics 的缩写。虽然其名称为虚拟内存统计，但它也可以完成 CPU、I/O 等方面的监控。vmstat 监控的是系统整体各个方面的指标，不能监控每个进程的具体情况。

1. 命令选项

vmstat 的命令格式如下：

```
vmstat [options] [delay [count]]
```

vmstat 命令在不加任何参数的情况下，默认输出的值为从系统启动到现在为止各个参数的平均值。count 参数为输出的次数，delay 参数为两次输出之间的间隔，单位为秒。如果只指定了 delay 参数而没有指定 count 参数，则会按照指定的延时时间一直输出。vmstat 命令常用的选项如表 1.4 所示。

表 1.4　vmstat 命令常用的选项

选　项	说　明	选　项	说　明
-a	显示 active/inactive 内存	-w	每列的宽度加宽
-m	显示 slabinfo 信息	-p partition	显示指定磁盘分区的统计数据
-d	显示磁盘统计数据	-S	指定输出的单位(k,K,m,M)

2. 示例

【例 1.4】 在命令行中输入不带任何参数的 vmstat 命令即可启动 vmstat 程序。其输出如下：

```
[root@localhost test]# vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------CPU-----
 r  b   swpd     free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0   127752     12    200    0    0     1     0  105    3 10  0 90  0  0
```

vmstat 的输出共分为 6 部分，分别为 procs、memory、swap、io、system、CPU，每部分的含义如表 1.5 所示。

表 1.5 vmstat 输出每部分的含义

分 组	字 段	含 义
procs	r	等待运行的进程数
	b	阻塞的进程数
memroy	swpd	已经使用虚拟内存的大小
	free	剩余的空闲内存
	buff	缓冲区占用内存大小
	cache	缓存占用内存大小
swap	si	每秒从 swap 分区交换到内存的大小，单位为 KB
	so	每秒从内存写入 swap 分区的大小，单位为 KB
io	bi	每秒从块设备接收到的块数
	bo	每秒发送到块设备的块数
system	in	系统每秒的中断，包含时钟中断
	cs	每秒上下文切换的次数
CPU	us	用户空间占用 CPU 的百分比
	sy	内核空间占用 CPU 的百分比
	id	空闲 CPU 的百分比
	wa	等待 I/O 的 CPU 占用百分比
	st	虚拟化消耗的 CPU 占比

【例 1.5】 运行 CPU 占用程序，使用 vmstat 进行监控。

使用命令 java CPUConsume 100 启动程序，同时运行 100 个线程占用 CPU。使用 vmstat 进行监控，以 MB 为单位显示，每秒采样一次，共采样 5 次。

```
[root@localhost ~]# vmstat -S M 1 5
procs -----------memory---------- ---swap-- -----io---- -system-- ------CPU-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
108  0      0 127498     18    405    0    0     0   106    0    4  3  0 95  1  0
101  0      0 127498     18    405    0    0     0    20 21715 2127 86  0  0  0 14
100  0      0 127498     18    405    0    0     0     0 23079 2183 91  0  0  0  9
100  0      0 127498     18    405    0    0     0     0 21530 1974 83  0  0  0 17
100  0      0 127498     18    405    0    0     0     0 21577 1956 85  0  0  0 15
```

以上 Java 程序启动了 100 个线程同时运行，通过对 vmstat 的结果分析得到如下结论。

r 列：5s 内当前系统等待运行的任务的数量为 100 左右，如果该值的输出长期大于系

统的逻辑 CPU 的数量，则表示等待的进程或线程数比较多，CPU 非常繁忙，此时 CPU 可能存在瓶颈。

b 列：系统当前阻塞的进程数为 0，当此值较大时，需要具体分析阻塞的进程，查看系统是否处于正常状态。

swpd 列：系统使用虚拟内存数为 0，如果 swpd 不为 0，则表示可能物理内存不足，但是也需要一并观察 si 和 so 两列是否同时有内存页面进行交换。如果系统虚拟内存使用大于 0，但是内存交换为 0，则可能是在某一时间系统内存不足，后期有占用内存较大的进程结束，释放内存后，依然有部分其他进程数据在交换分区中。如果 swpd 使用较大，同时 si、so 交换频繁，则表示系统当前状态内存不足。

free 列：系统当前剩余的物理内存大小为 127GB 左右，如果系统频繁地使用交换分区，往往其 free 的值也剩余无几。

in 列：系统每秒中断的数量为 22 000 左右。

cs 列：系统每秒上下文切换的次数为 2000 左右，如果此值过高，则说明系统上下文切换浪费了很多 CPU 资源，同时往往伴随着 r 列等待运行的进程数较多。因此这个数值应越小越好。

us 列：用户空间 CPU 时间占比为 90%左右，用户空间占比越大说明 CPU 在用户进程消耗的时间越多，因此此值应较大为宜。

sy 列：内核空间 CPU 时间占比为 0，一般情况下，该值不宜过大。

【例 1.6】 运行磁盘 I/O 占用程序，使用 vmstat 进行监控。

使用命令 java DiskIOConsume 100 启动程序，同时运行 100 个线程向磁盘写入数据。使用 vmstat 进行监控，以 MB 为单位显示，每秒采样一次，共采样 5 次。

```
[root@localhost ~]# vmstat -S M 1 5
procs -----------memory---------- ---swap-- -----io---- -system-- ------CPU-----
r  b   swpd    free   buff   cache   si   so    bi     bo    in    cs  us sy id  wa  st
1 101      0 127138     18     496    0    0     0    108     6     4   4  0 95   1   0
0 101      0 127139     18     494    0    0     0 199208 5354  1926   1  4  0 95   0
0 101      0 127139     18     493    0    0     0 168896 4771  1791   1  3  0 96   0
0 101      0 127138     18     493    0    0     0 191488 4903  1655   1  4  0 95   0
0 101      0 127132     18     501    0    0     0 193408 4744  1629   1  4  2 93   1
```

以上 Java 程序启动了 100 个线程向磁盘写入数据，通过对 vmstat 的结果分析可知：b 列显示当前系统阻塞任务数为 101，同时 bo 列显示磁盘写入速率为 190MB/s 左右，wa 显示当前 CPU 在 I/O 等待上消耗为 95%左右。由此可见，当前系统正在大量写入数据，100 多个任务阻塞，大量 CPU 资源浪费在等待 I/O 的操作上，此时可判定系统当前状态磁盘压力过大。

1.1.5　iostat 命令

虽然 vmstat 可以查看到磁盘的 I/O 情况，但是其显示的信息，如系统的哪块磁盘读写较高、等待 I/O 队列长度等却不是很详细。iostat 是一个专门分析磁盘 I/O 的工具，可以查看各种磁盘 I/O 的详细参数。

1. 软件安装

默认 CentOS 7 发行版中没有安装此工具，使用以下命令进行安装：

```
yum install sysstat - y
```

安装完成后，输入 iostat 命令，检查是否安装成功。

```
[root@localhost ~]# iostat
Linux 3.10.0 - 693.el7.x86_64 (localhost)        12/05/2018     _x86_64_       (24 CPU)

avg - CPU:   % user    % nice  % system % iowait   % steal    % idle
             3.77      0.00     0.05     1.51      0.16     94.51

Device:             tps     kB_read/s     kB_wrtn/s     kB_read     kB_wrtn
vda                9.11          6.05       2987.73      289989   143290900
```

2. 命令选项

iostat 命令格式如下：

```
iostat [options] [delay [count]]
```

iostat 命令在不加任何参数的情况下，默认输出的值为从系统启动到现在为止各个参数的平均值。count 参数为输出的次数，delay 参数为两次输出之间的间隔，单位为秒。如果只指定了 delay 参数而没有指定 count 参数，则会按照指定的延时时间一直输出。此用法与 vmstat 类似。iostat 命令常用的选项如表 1.6 所示。

表 1.6　iostat 命令常用的选项

选　　项	说　　明
-c	只显示 CPU 状态信息
-d	只显示磁盘状态信息
-k	强制使用 KB 为单位
-x	显示更详细的信息
-p device	显示指定设备的信息
-y	如果多次显示，不显示第一次而显示从系统启动后的平均值

3. 示例

【例 1.7】 在命令行中输入 iostat -x 命令启动 iostat 程序。其输出如下：

```
[root@localhost ~]# iostat - x
Linux 3.10.0 - 693.el7.x86_64 (localhost)        12/05/2018     _x86_64_       (24 CPU)

avg - CPU:   % user    % nice % system % iowait   % steal    % idle
             3.72      0.00     0.05     1.49      0.16     94.58

Device:      rrqm/s    wrqm/s     r/s     w/s     rkB/s     wkB/s avgrq - sz avgqu - sz
await   r_await   w_await   svctm   % util
vda            0.00      0.15     0.14     8.84     5.96   2946.38    657.49      2.01
222.45     2.35   225.84    2.03    1.83
```

在输出中 avg-CPU 一行输出的为多核 CPU 的平均信息，其内容与 top 中的 CPU 输出信息类似。

下半部分显示每一个块设备或分区的详细的读写信息。每一列的含义如下。

- Device：设备名称或分区名称。
- rrqm/s：发送到设备队列每秒合并的读请求数。
- wrqm/s：发送到设备队列每秒合并的写请求数。
- r/s：每秒完成的合并后的读请求数。
- w/s：每秒完成的合并后的写请求数。
- rkB/s：每秒读取的数据量。
- wkB/s：每秒写入的数据量。
- avgrq-sz：每个请求的平均扇区数。
- avgqu-sz：平均请求队列长度。
- await：平均每次读写所需的时间。
- r_await：平均每次读需要的时间。
- w_await：平均每次写需要的时间。
- svctm：废弃指标。
- util：读写时间占总时间的百分比。

【例 1.8】 运行磁盘 I/O 占用程序，使用 iostat 进行监控。

使用命令 java DiskIOConsume 100 启动程序，同时运行 100 个线程向磁盘写入数据。使用 iostat 进行监控，以 MB 为单位显示，每秒采样一次，共采样 3 次。

```
[root@localhost ~]# iostat  -myx 1 3
Linux 3.10.0-693.el7.x86_64 (localhost)     12/20/2018     _x86_64_       (24 CPU)

avg-CPU:   %user    %nice %system %iowait   %steal    %idle
            0.54     0.00    4.14   95.07     0.25     0.00

Device:     rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz
await   r_await w_await   svctm   %util
vda           0.00     0.00    0.00  555.00     0.00   184.52   680.91   127.92
258.40    0.00  258.40    1.80  100.10

...
```

以上 Java 程序启动了 100 个线程向磁盘写入数据，通过对 iostat 的结果分析可知：当前系统 CPU 的 I/O 等待占比为 95.07%，大量时间浪费在等待 I/O 的操作上。此时对系统的性能影响比较严重。系统当前 vda 设备每秒处理 555 次合并后的写请求，写入速度为 184.52MB/s，平均队列长度为 127.92，I/O 等待时间为 258.40ms，I/O 时间占比为 100.10%，由此可见系统当前 I/O 特别繁忙，I/O 队列长度较长，每次 I/O 等待时间也较长。如果系统长期处于这种状态，则可能需要考虑提高 I/O 设备性能或优化程序性能。

1.1.6　iftop 命令

iftop 是一个网卡流量的实时监控工具。通过 iftop 可以查看到当前系统网卡的实时流入流出流量。iftop 还可以监控当前系统与外界系统通信的流量情况，清晰直观地显示每一

个外界系统流入流出的速率、IP 地址等信息。使用 iftop 命令对于定位系统中的异常流量问题是十分简单有效的。

1. 软件安装

默认 CentOS 7 发行版本中没有安装此工具，安装 iftop 需要使用 EPEL 源，使用以下命令进行安装：

```
yum install iftop - y
```

安装完成后，输入 iftop 命令，检查是否安装成功。

2. 命令选项

iftop 命令格式如下：

```
iftop [options]
```

iftop 命令在不加任何参数的情况下，默认显示第一块网卡的流量信息。iftop 命令常用的选项如表 1.7 所示。

表 1.7　iftop 命令常用的选项

选　项	说　　明	选　项	说　　明
-i	指定需要监控的网卡	-P	显示端口号
-B	以 B 为单位显示，默认显示的单位为 b	-n	显示 IP 地址，不进行 DNS 反向解析
-F	显示特定网段的网卡流量信息	-N	只显示端口号，而不显示服务名。如 ssh 显示为 22
-m	调节顶部刻度的最大值，如 iftop -m 100m		

3. 示例

【例 1.9】 在命令行中输入 iftop -nNP -m 20m 命令启动 iftop 程序。其输出如图 1.3 所示。

```
                 4.00Mb              8.00Mb              12.0Mb              16.0Mb              20.0Mb
└──────────────────┴──────────────────┴──────────────────┴──────────────────┴──────────────────
192.168.111.41:6801                 => 192.168.111.50:49428                8.53Kb  8.53Kb  8.53Kb
                                    <=                                     8.53Kb  8.53Kb  8.53Kb
192.168.111.41:6801                 => 192.168.111.43:33360                8.53Kb  8.53Kb  5.33Kb
                                    <=                                     8.53Kb  8.53Kb  5.33Kb
192.168.111.41:45034                => 192.168.111.45:6800                    0b   5.58Kb  6.53Kb
                                    <=                                        0b   10.0Kb  6.48Kb
192.168.111.41:6800                 => 192.168.111.54:37650                   0b   9.97Kb  12.2Kb
                                    <=                                        0b   5.46Kb  3.78Kb
192.168.111.41:6801                 => 192.168.111.54:54688                   0b   6.82Kb  6.40Kb
                                    <=                                        0b   6.82Kb  6.40Kb
192.168.111.41:6801                 => 192.168.111.46:41492                8.53Kb  6.82Kb  5.33Kb
                                    <=                                     8.53Kb  6.82Kb  5.33Kb
192.168.111.41:6800                 => 192.168.111.49:43008                   0b   9.70Kb  12.1Kb
                                    <=                                        0b    645b   806b
192.168.111.41:6801                 => 192.168.111.52:47580                8.53Kb  5.12Kb  7.46Kb
                                    <=                                     8.53Kb  5.12Kb  7.46Kb
192.168.111.41:6801                 => 192.168.111.49:60816                   0b   5.12Kb  6.40Kb
                                    <=                                        0b   5.12Kb  6.40Kb
192.168.111.41:6801                 => 192.168.111.56:57036                8.53Kb  5.12Kb  5.33Kb
                                    <=                                     8.53Kb  5.12Kb  5.33Kb

──────────────────────────────────────────────────────────────────────────────────────────────
TX:             cum:   329KB   peak:   352Kb              rates:    205Kb   159Kb   165Kb
RX:                    315KB           262Kb                        208Kb   156Kb   157Kb
TOTAL:                 644KB           614Kb                        413Kb   314Kb   322Kb
```

图 1.3　iftop 输出界面

输出共分为 3 部分。顶部为刻度条，是中间部分和底部图形部分的参考刻度。

中间部分为系统与外界连接的信息，包括 IP 地址、端口和系统在 2s、10s、40s 内的数据传输速率。其中=>为发送数据，<=为接收数据。

底部内容分为 3 行，即 TX、RX、TOTAL，分别表示数据的发送、接收和全部的流量数据。cum 列为从运行 iftop 以后总共的流量数据。如在 TX 行中，cum 列运行 iftop 以后，总共发送数据 329KB。peak 为传输速率的峰值。rates 为 2s、10s、40s 内的数据传输速率。

1.2　Prometheus

1.1 节中介绍了 Linux 中常见的性能监控工具，但是这些工具都是监控单台机器的某些特定的指标，无法从整体看到某台机器的运行状态，也无法直观地以图表形式进行展示。本节介绍 Prometheus 监控系统，实现对机器的各项指标的监测，并以丰富的图形界面形式进行展示。

1.2.1　Prometheus 简介

Prometheus 是一个开源的系统监控及报警工具，自 2012 年推出以来，许多公司都采用它搭建监控和报警系统。它拥有非常活跃的开发人员和用户社区，在 2016 年加入了云原生计算基金会(CNCF)，成为继 Kubernetes 之后的第二个托管项目。

1. Prometheus 的特点

- 使用一系列由键值对组成的时间序列多维数据模型。
- 使用灵活的 PromQL 查询语言。
- 不依赖分布式存储系统，单节点自治。
- 通过 HTTP 的方式拉取时间序列数据。
- 支持向中间网关主动推送数据。
- 可通过服务发现或配置方式发现监控目标。
- 以多种图形方式或仪表盘的形式进行展现。

2. 适用场景

Prometheus 适用于任何纯数字的时间序列数据，既适用于以机器为中心的监控，也适用于动态的面向服务的监控。Prometheus 可以对各种服务进行监控，如 SpringBoot、Spark。在微服务的监控中，其多维数据的收集和查询成为一种特殊的优势。Prometheus 不适合要求数据 100%准确的场景，如作为计费系统使用，因为它收集的数据可能不够完整。

1.2.2　Prometheus 的组成

Prometheus 生态系统由很多组件组成，最核心的组件为 Prometheus server，其架构如图 1.4 所示。

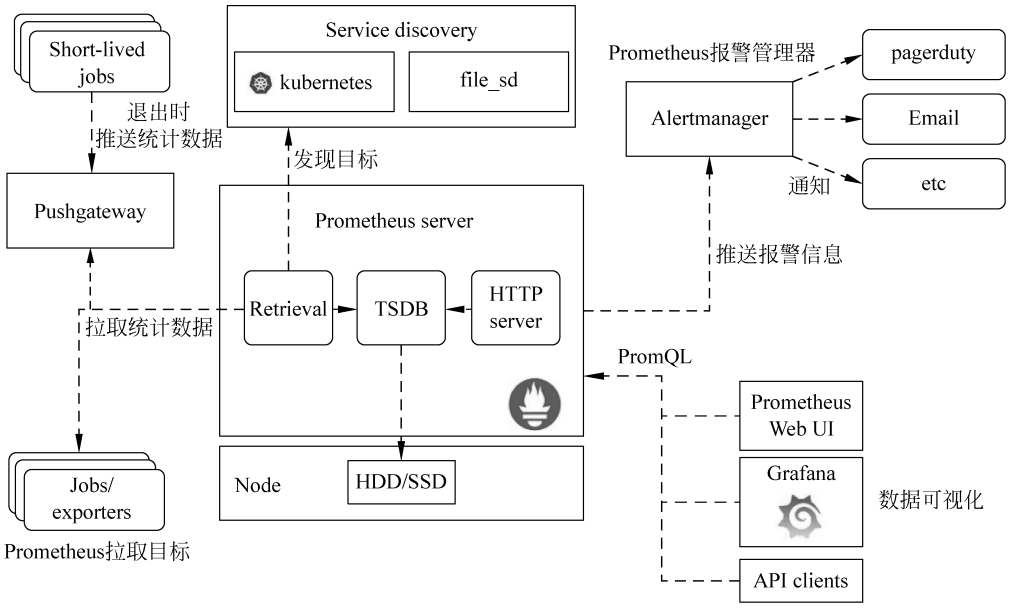

图 1.4　Prometheus 架构

Prometheus server 负责定时的拉取时间序列数据，并将数据进行保存，对外提供 HTTP 接口供其他程序查询。

Exproter 为一个应用程序，它将采集到的信息转换为 Prometheus 支持的格式，并对外提供接口，供 Prometheus server 拉取数据。监控不同类型的数据需要使用不同的 Exporter，如监控服务器性能指标可以使用 node_exporter，监控 Java 程序可以使用 JMX exporter。Prometheus 官方维护了部分常用的 Exporter，同时有很多大量的第三方 Exporter 用于监控各种不同的应用程序。常用的 Exporter 列表可以参考官方文档，地址为 https://prometheus.io/docs/instrumenting/exporters/。

Prometheus 不仅可以主动拉取监控数据，还允许应用程序主动将监控数据进行推送，但是应用程序并不是直接将数据推送至 Prometheus server 中，而是推送至中间的网关称为 Pushgateway。有些服务级别的应用程序可能运行时间很短，还没等 Prometheus server 来拉取数据，程序就结束了。在这种情况下，应用程序在结束的时候可以将数据推送至 Pushgateway，Prometheus server 再从 Pushgateway 中拉取数据，进而获取短时间运行任务的统计数据。

Prometheus 将采集后的数据，使用一些可视化工具通过 PromQL 便可以对数据进行查询。如 Prometheus 自带的 Web UI，还可以使用更强大的 Grafana 等对数据进行可视化的展示。

Prometheus 除了可以对各种指标进行监测外，还可以实现报警的功能。通过编写特定的规则，当 Prometheus 采集到的数据满足规则时，便可以触发报警。Prometheus server 本身并不提供报警的实现，其报警功能由 Altermanager 组件实现。Prometheus server 把满足规则的报警事件推送给 Altermanager，Altermanager 将报警事件做后续处理，如发送邮件、推送至其他平台等。

1.2.3　Prometheus 的安装及配置

1. 安装

Prometheus 安装比较简单，官方提供了已发布的二进制程序包，直接解压即可使用。下载地址为 https://prometheus.io/download/。

将下载的安装包解压，并切换至软件目录：

```
wget https://github.com/.../prometheus-2.6.0.linux-amd64.tar.gz
tar xvfz prometheus-*.tar.gz
cd prometheus-*
```

其目录下的 prometheus 即为可执行的二进制程序，prometheus.yml 为配置文件。使用./prometheus 即可加载当前目录下的配置文件，启动 Prometheus。

2. 配置

打开 prometheus.yml 文件，其内容结构如下：

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093
rule_files:
  # - "first_rules.yml"
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
    - targets: ['localhost:9090']
```

prometheus.yml 文件整体分为 4 部分内容。global 为全局配置，为其他的配置部分提供了默认值。如 scrape_interval 表示每隔多长时间拉取一次数据；evaluation_interval 表示每隔多长时间对数据按照 rules 进行一次校验，查看是否有报警事件发生。

alerting 配置部分为报警的配置，配置 Altermanager 的地址，当 Prometheus server 检查到有报警事件时，会根据 alterting 的配置，将事件推送到相应的 Altermanager 中。

rule_files 配置相应的报警规则，每个规则可以指定一个单独的文件，其报警检查可以检查多个规则，一旦事件符合某个规则，则将事件推送至 Altermanager 中。

scrape_configs 为最重要的部分，是拉取数据的配置部分。一个 Prometheus server 可以配置多个拉取的任务，每个任务被称为一个 Job。通常一个 Job 由一组功能相同的 target 组成。如一个 Job 为监控所有的服务器指标，另一个 Job 为监控所有的 Spark 任务的指标。每个 Job 在配置文件中通过 job_name 区分，job_name 必须是唯一的。static_configs 用于配置当前 Job 的固定值，如拉取数据的地址、附加的标签等。targets 配置可以指定一组需

要拉取数据的地址，这些地址都是 exporter 对外提供的接口的地址。

3. 启动

使用./prometheus 命令启动 Prometheus server。Prometheus server 启动后，使用 9090 端口作为 Web UI 的 HTTP 服务的端口。如果该机器地址为 192.168.99.6，则使用浏览器访问地址 http://192.168.99.6:9090 查看 Prometheus server 是否正常启动。

1.2.4 监测服务器

1. 安装 node_exporter

实现对服务器指标的监测需要使用采集服务器指标的 Exporter，这个 Exporter 被称为 node_exporter。官方的下载地址为 https://prometheus.io/download/。下载完成后，将文件解压，直接运行其二进制文件 node_exporter 即可启动，其命令如下：

```
wget https://... /download/v0.17.0/node_exporter-0.17.0.linux-amd64.tar.gz
tar-zxvf node_exporter-0.17.0.linux-amd64.tar.gz
cd node_exporter-0.17.0.linux-amd64
./node_exporter
```

假如该机器的 IP 地址为 192.168.99.6，node_exporter 启动以后，默认使用本机的 9100 端口提供 HTTP 服务。使用浏览器访问地址 http://192.168.99.6:9100/metrics，可查看启动是否正常，是否返回数据。使用 curl 访问该地址返回数据如下：

```
[root@locahost]# curl http://localhost:9100/metrics |  head-20
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0.000188284
go_gc_duration_seconds{quantile="0.25"} 0.000188284
go_gc_duration_seconds{quantile="0.5"} 0.000305924
...
```

2. 配置 Prometheus server

node_exporter 安装完成后以后，即可配置 Prometheus server，将 targets 配置为 node_exporter 的地址。其部分配置如下：

```
scrape_configs:
  - job_name: 'node-status'
    static_configs:
    - targets:
      - 192.168.99.6:9100
```

按照以上配置完成后，启动 Prometheus server，Prometheus 即可定时地到 http://192.168.6.9100/metrics 地址中拉取 node_exporter 采集到的数据。此外如果同一个 Job 中不同的机器拥有不同的属性，也可以使用 labels 添加自定义的属性。如在一个 Job 中都是监控的服务器的状态，而在服务器中有的为测试环境机器，有的为生产环境机器，使用 labels 标签可以进行区分。其实例配置如下：

```
scrape_configs:
  - job_name: 'node - status'
    static_configs:
    - targets:
      - 192.168.99.6:9100
      labels:
        env: test
    - targets:
      - 192.168.100.6:9100
      labels:
        env: product
```

3. 查看监测指标

Prometheus server 的 Web UI 界面提供了简单的指标查询功能，访问地址 http://192.168.99.6:9090，选择指标为 process_CPU_seconds_total，打开 Graph 选项卡，查看显示是否正常。其界面如图 1.5 所示。

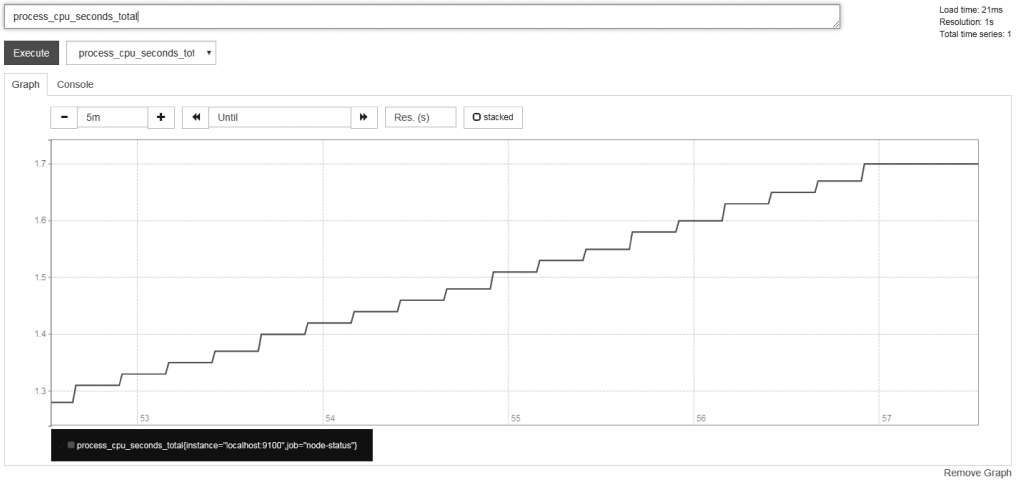

图 1.5　Prometheus Web UI

1.3　Grafana

Prometheus 完成了监测数据的采集、存储、报警等功能，但其对于采集到的指标在可视化方面显得有些单调，只能以表格或图标的形式进行简单的展示，对于重要的指标也不能够自定义面板显示。本节介绍 Grafana 的使用，通过 Grafana 实现后续的可视化展示。

1.3.1　Grafana 简介

Grafana 是一个开源的度量分析和可视化的工具。其主要特点如下。

1. 可视化

Grafana 拥有快速灵活的多选项的图表，可通过面板插件使用不同的方式实现对各种

统计数据或日志进行可视化。

2. 报警

Grafana 可以通过可视化的方式为重要的指标配置报警规则，它将自动地监控这些指标并在指标异常的时候发送通知。

3. 数据源统一

Grafana 支持数十种数据库数据源，可以将各种数据整合在同一个平台中，在同一个面板中进行展示。

4. 扩展性

Grafana 拥有数百个面板和插件库。这些插件库几乎可以完成各种常见数据的展示，并且该插件库还在不断地更新，添加新的功能。

Grafana 通过接入不同类型的数据源，在其界面中可以配置可视化面板，通过编写相应的查询语句，配置不同的面板，实现不同类型的数据展示，如折线图、饼状图、表格等方式。Grafana 实现的功能如图 1.6 所示。

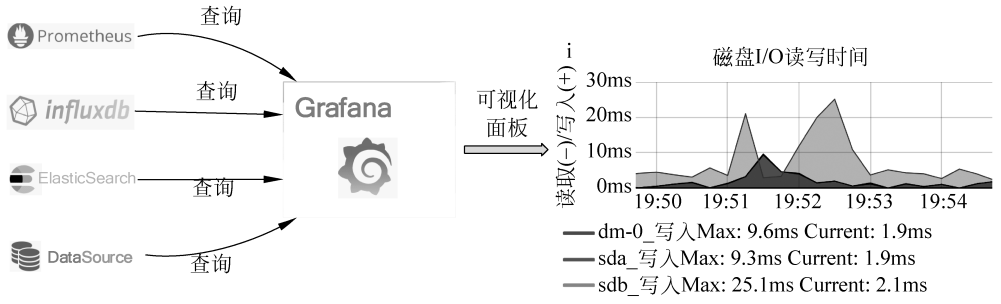

图 1.6　Grafana 功能示意图

1.3.2　Grafana 的安装

1. 网络安装

Grafana 官方提供 rpm 包，可以直接在 CentOS 中安装使用。可使用以下命令直接通过网络进行安装：

```
yum install https://s3-us-west-2.amazonaws.com/grafana-releases/release/grafana-5.1.4-1.x86_64.rpm
```

2. yum 源安装

官方也提供了 yum 源，可通过 yum 源进行安装。在系统中创建文件/etc/yum.repos.d/grafana.repo，文件中写入 yum 源的地址，内容如下：

```
[grafana]
name=grafana
baseurl=https://packagecloud.io/grafana/stable/el/7/$basearch
repo_gpgcheck=0
```