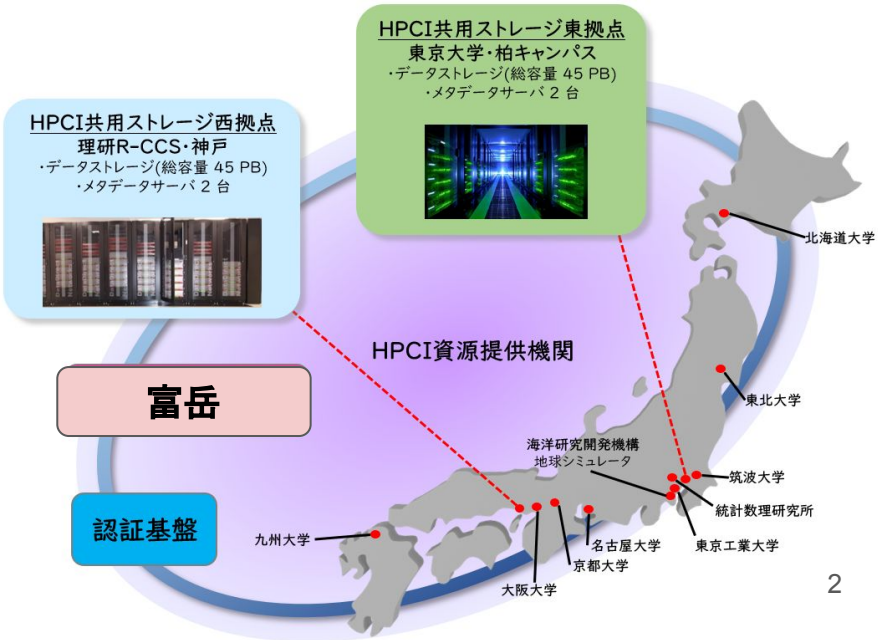
HPCI共用ストレージ利用者向け
情報公開サービス の構築と展望

2021/03/05(Fri)

R-CCS　金山 秀智
原田 浩
芝野 千尋

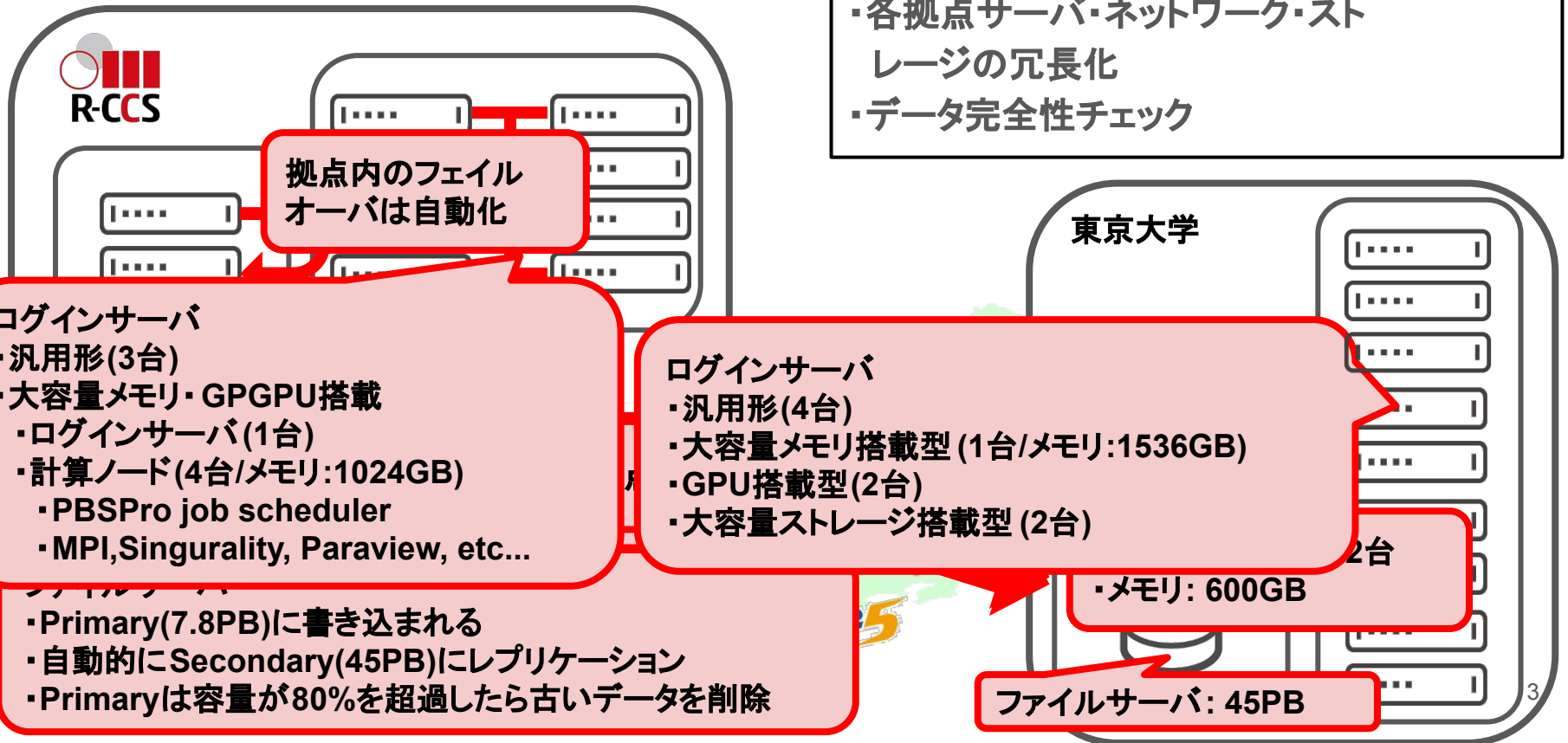
HPCI共用ストレージの紹介

- HPCI の資源としてストレージ資源を提供
- ファイルシステムGfarmを利用
- 全国のHPCI計算機資源から、高速・大容量の単一ファイルシステムとしてデータ共有が可能
- 容量: 論理45PB (物理90PB)
  → 拠点間多重化
- 性能:
  - 100Gbpsの対外ネットワーク帯域
  - 最大100GbpsでのファイルI/O
- 特徴
  - シングルサインオンで全国のスパコンセンターから利用可能



HPCI共用ストレージの構成

・東大・R-CCS拠点の片拠点運用可能
　・拠点間多重化
・各拠点サーバ・ネットワーク・ストレージの冗長化
・データ完全性チェック



発表内容

2021年01月29日。HPCI共用ストレージでは「サービス稼働情報」と「利用者向け情報」の公開を開始しました。

URL: https://hpci-web01.r-ccs.riken.jp/grafana

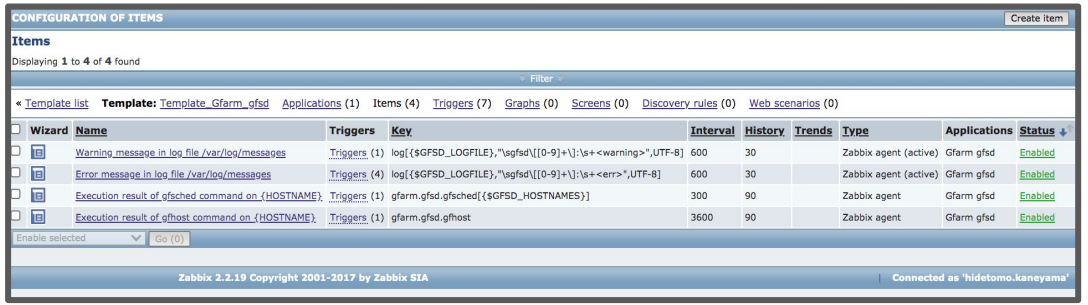
Manual: https://www.hpci-office.jp/info/pages/viewpage.action?pageId=216629492

設計・現在の状況・展望について発表します！

運用向けの障害監視はGfarm Zabbixを利用

Gfarmでは障害監視用のZabbixプラグイン Gfarm Zabbix が整備

- Gfarmのアラートチェック
  - <err>,<warning>, I/Oエラーの監視...
- PostgreSQL(メタDB)の死活監視
- データ改善性チェックの状態監視
- gfmd,gfsdの説状況確認
- フェイルオーバの実行　などなど



これに情報の取得や運用者向けに一部の情報を可視化、ハードウェア障害検知などをZabbixに追加監視を強化

- 二拠点拠点間の死活監視状況
- ハードウェアアラート
- ネットワーク帯域
- gfstatusで取得可能な設定状況
- 情報グラフ などなど

→ Gfarmの運用に必須な監視項目はGfarm Zabbix を利用
→ 共用ストレージでは運用開始時点から導入
→ 東大・R-CCS個別に準備して運用・監視に利用

サービスを監視したい/ユーザに情報を提供したい

クラウドサービスでは様々なサービス情報を[illegible]に GUIで
(利用量/疎通状態/利用可能な[illegible]

共用ストレージでは…
・障害が発生した場合のみユーザ[illegible]
・利用状況をメールで実施

メリットいっぱい

[illegible]の提供情報を増やしたい！

→ ユーザに最新のサービス運用情報や利用情報を提供したい　/ 利便性の強化・ユーザ利用難易度の削減
→ 共用ストレージをユーザが実際に利用できるかを把握したい　/ 監視の正確性向上・ユーザ目線での監視
→ 運用メンバの手を介さずに自動的にユーザに情報を提供したい　/ 運用コスト削減
→ 一目で障害状況や障害規模が把握できるようにしたい　/ 運用難易度の削減

監視機能を強化して公開する！！

- gfmdhostコマンドのレスポンス時
- available replicasログの確認
- ファイルサーバの帯域
- 書き込み・読み出しプロセス数
- 書き込み・読み出しユーザ数
- ユーザディレクトリのパーミッション
- グループディレクトリパーミッション
- ホストグループごとのReadOnly設定状況
- 利用量
- ファイル数
- ディレクトリ・シンボリックリンク数
- 接続gfmd数
- 接続gfsd数
- ホストグループごとの接続fsd数
- ファイルの書き込みができるか
- ファイルの読み出しができるか
- マウントができるか
- アンマウントができるか
- レプリカ数設定ファイル数
- ログインノードの接続数

ほとんどがGfarmのコマンドで取得できる！
ユーザにコマンドを説明するより可視化してWebページを見てもらう方が楽！

監視・ダッシュボードツールの選定

→ Prometheus + Grafanaでの構築を選択

- ユーザへの公開を目標
  - レイアウトが大変なので可視化部分はダッシュボードソフトと連携したい
  - Datadog,Mackerelなどの監視サービスとの連携も視野
- Gfarm以外のサービスも対象にしたい
- Zabbix運用で大変だった部分を改善したい
  - Zabbixのバージョンアップで苦労した記憶 ....(mysql,templateやり直し)
  - Zabbix負荷アップでの苦労 ...
- 複雑な文字列処理の苦労 ...
  - Zabbixはなんでもできちゃうので、複雑になる /DBが大きくなりがち
  - logなどはElasticSearchなどのログ監視サービスで
  - かといって連携したい ....
- 高メトリクス
  - 研究やテストで高頻度情報が必要になることがあるので
- Defact Standerd(情報収集量の観点から ...トレンドを抑えたい)
- コンテナやBPFとの連携(今後に備えて)

今後も使うので新しいもので(しばらく)廃れないものにしておきたい！
後々を考えてクラウドとの連携がしやすいもの
Zabbixの新バージョン or Prometheusで悩む(Sensuでもよかったかも)
ノウハウが多くダッシュボードソフト(Ganglia)やDatadog/Mackerelと相性が良いPrometheusを選択！

構築

Shibboleth認証(HPCIユーザ向け)
※ 今後ユーザ別情報の提供のため

WEBサーバ(proxy)
外部へ接続して公開

ダッシュボードソフト+DB

監視ソフト・情報格納・バックアップ

サービス監視環境は全てR-CCS無停電室に設置(停電などの影響を極力減らす)

ZABBIX

UTokyo Hub

R-CCS HUB

R-CCS

Shibboleth

APACHE HTTP SERVER PROJECT

MySQL

Grafana

Prometheus

influxdb

構築

Shibboleth

APACHE HTTP SERVER PROJECT

Grafana

MySQL

ZABBIX

Prometheus

influxdb

調査などで後で何の情報が必要になるかわからないので取得可能な情報は取得しておく

サービス専用のExporterをインストールしてGfarm以外も監視強化

可視化はGrafanaにお任せ

バックアップ

監視対象

Gfarm運用情報

ZABBIX Zabbix Agent & gfarm Zabbix

機材情報

ZABBIX Zabbix Agent

Prometheus

クライアント環境からコマンドを実行して、サービスの状態を監視

その他の運用情報(Ex: PBS,

Prometheus Prometheus Exporter

テコ入れ

サービス情報

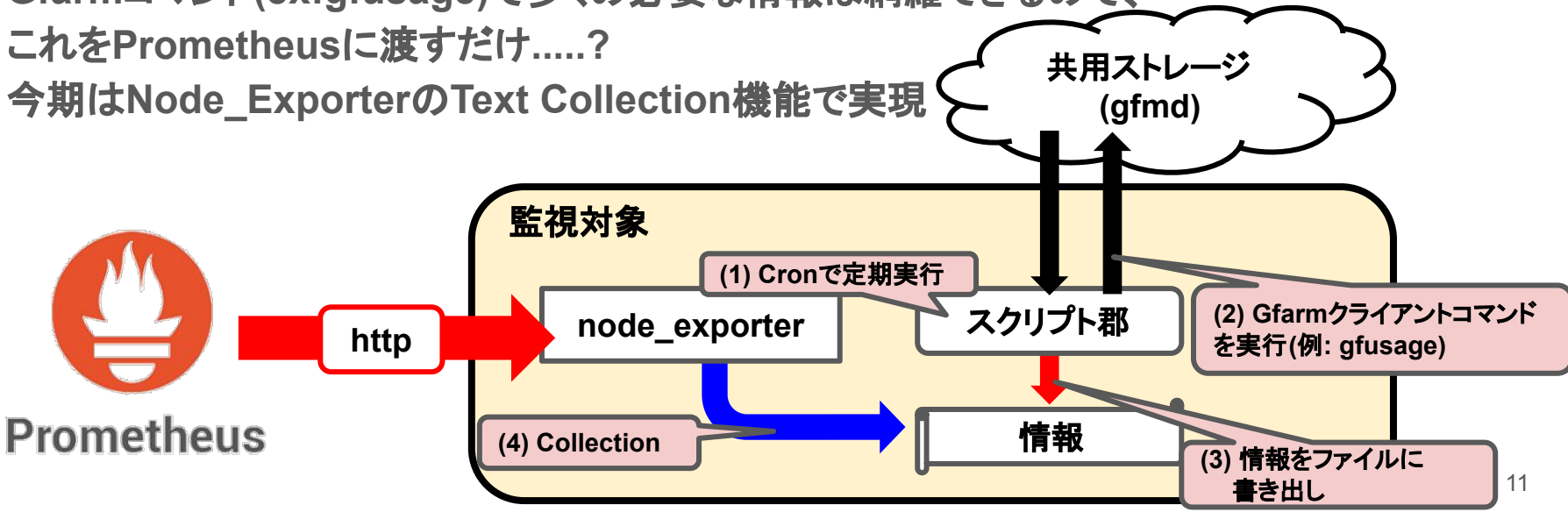
Prometheus Exporter & Scripts

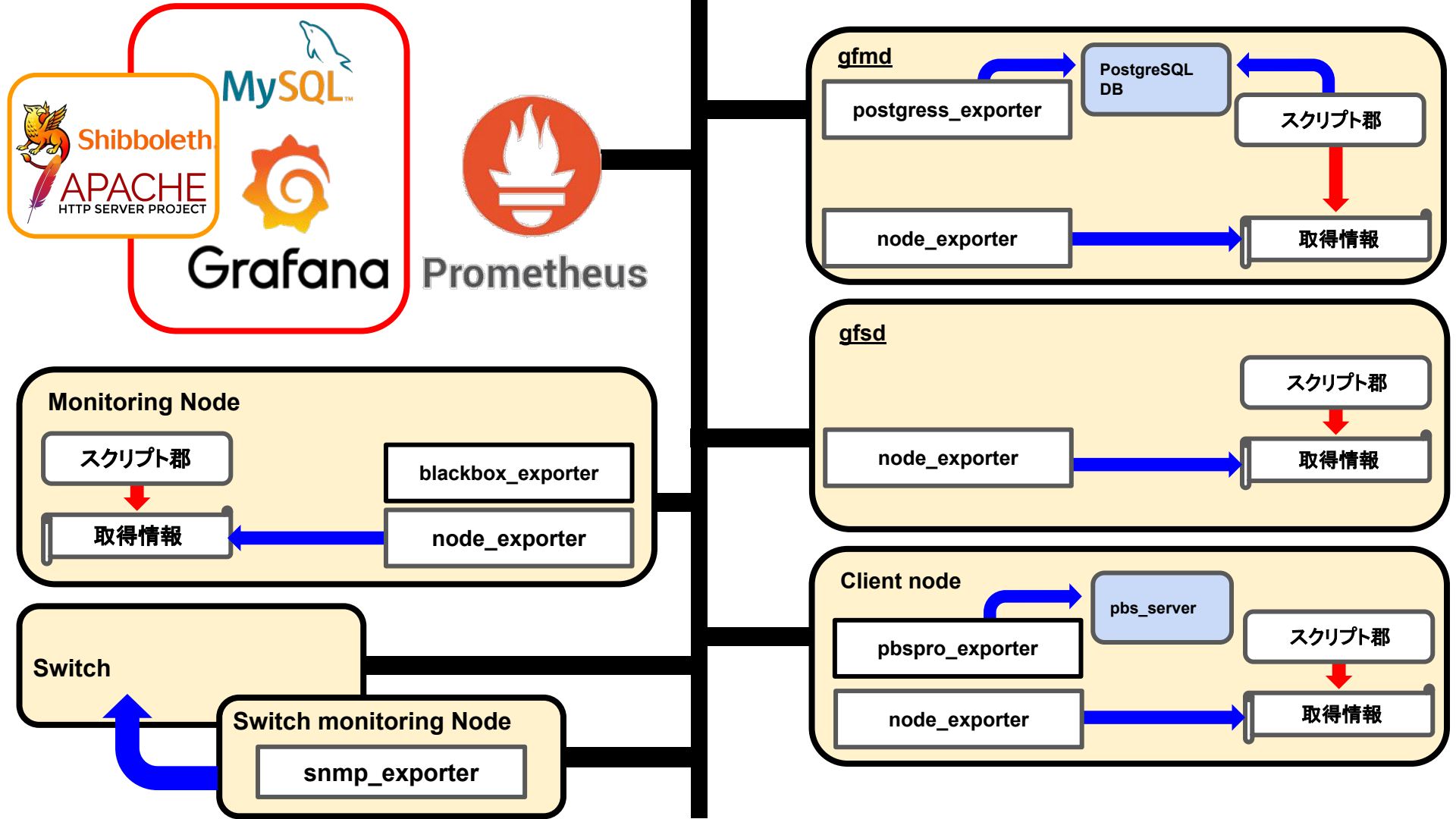
Exporter

Prometheusが情報を収集するために監視対象にインストールするツール
情報を収集し、結果をPrometheusが読み出せる形式にして提供
監視サービスや対象ごとにインストールするExporterは異なる
種類は豊富、自作も可能(PBS/Apache/Nginx/Postgres/NVIDIA GPU…)

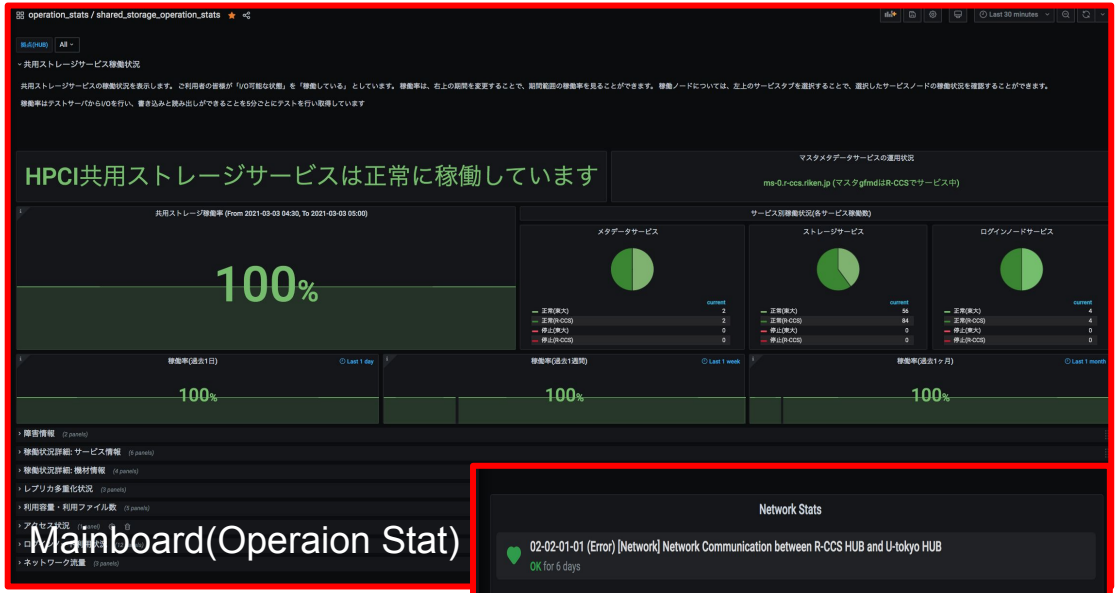
サービス監視の実装

- 共用ストレージはGfarmをファイルシステムとして採用している
- サービス状況は利用情報は、
  ・Gfarmが正常に動作しているかを確認！
  ・Gfarm(gfmd)から情報を取得！
- Gfarmコマンド(ex:gfusage)で多くの必要な情報は網羅できるので、
  これをPrometheusに渡すだけ.....?
- 今期はNode_ExporterのText Collection機能で実現

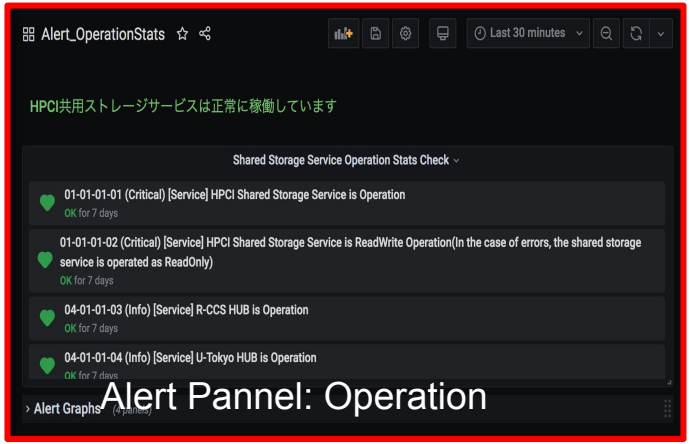



Shibboleth
APACHE
HTTP SERVER PROJECT
MySQL
Grafana
Prometheus
gfmd
postgress_exporter
PostgreSQL
DB
スクリプト郡
node_exporter
取得情報
gfsd
スクリプト郡
node_exporter
取得情報
Monitoring Node
スクリプト郡
取得情報
blackbox_exporter
node_exporter
Client node
pbs_server
pbspro_exporter
スクリプト郡
node_exporter
取得情報
Switch
Switch monitoring Node
snmp_exporter

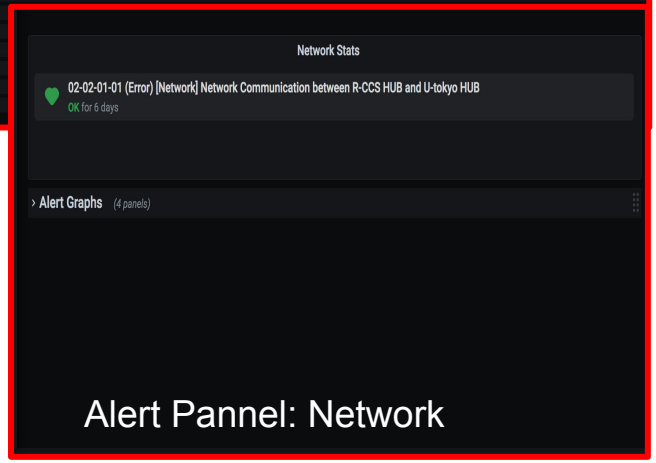
DEMO: https://hpci-web01.r-ccs.riken.jp/grafana/d/SS_OPERATION/shared_storage_operation_stats?orgId=2



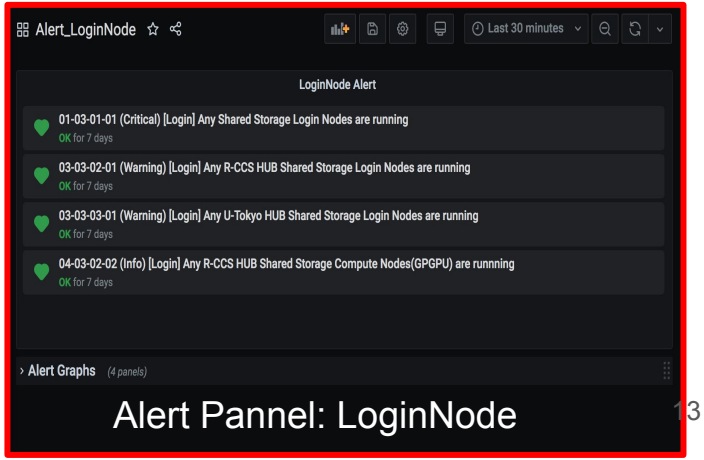
Mainboard(Operaion Stat)



Alert Pannel: Operation

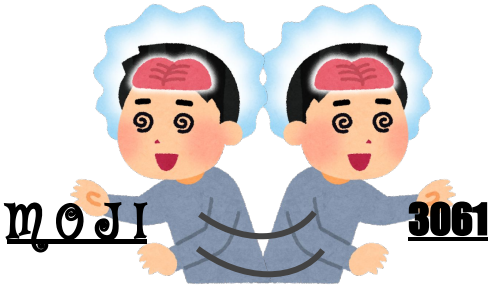


Alert Pannel: Network



Alert Pannel: LoginNode

苦労話/文字処理は手間



GfarmコマンドからPrometheusへ情報をを渡すには…
→ コマンドの情報は数値ではなく文字列
→ Prometheusは数値受け取りが基本

文字処理が必要・面倒

雑記
- SNMPのPacketカウンタのような情報だと管理しやすい
- メタDBでSQLを叩いて情報を取得するような方法が良いのかも知れない…
- これだと、ユーザコマンドを作っているのと同じなのでは
- (Exporterを開発する際はAPIを直接叩いて情報取得する方が良いのかな…)
- (Power Shellみたいにobjectだったり、csvフォーマット,JSONなら良いのになー)

```
$ gfusage hpci002329
#   UserName :       FileSpace      FileNum
                     PhysicalSpace PhysicalNum
  hpci002329 :  50263739564644      3353910
                252684734623281     15074613
```

VS

```
gfarm=> SELECT SUM(size) FROM inode WHERE
            username='hpci002329' AND mode!=0;
      sum
----------------
 50263739564644
```

メタデータDBの活用

メタDBから直接取得が必要な情報もあった

- ディレクトリ数やシンボリックリンク数
- レプリカ数指定されたファイル数
- レプリカ作成状況などなど

(例): レプリカ数・ホストグループ指定されたinode一覧の取得方法

```
SELECT DISTINCT inumber FROM xattr WHERE attrname in ('gfarm.replicainfo','gfarm.ncopy')
```

Gfarmコマンドだけでは監視できない情報はメタデータから取得
→ このような情報の取得にはGfarmのメタDBのデータ構造への理解が必要
→ 取得内容に漏れがないか、取得可能な情報を見逃していないか

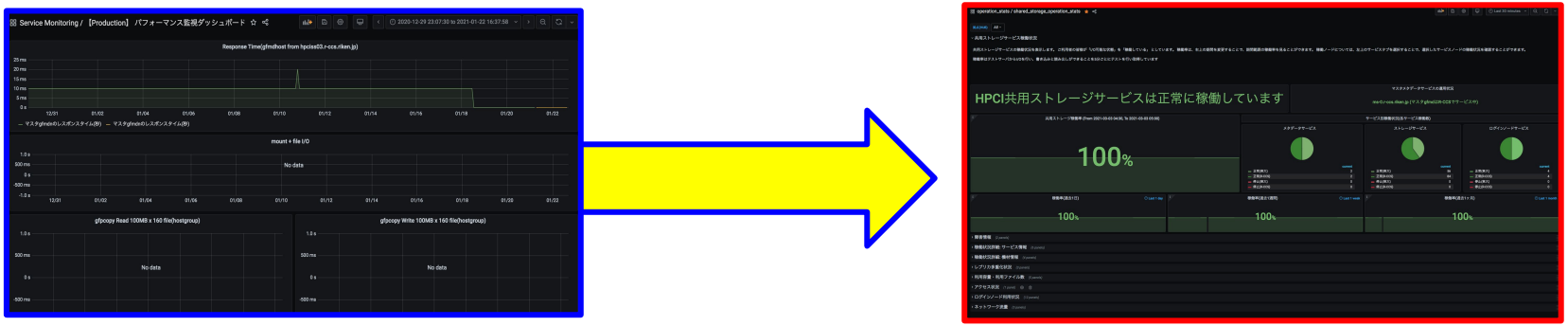
ダッシュボード作りは大変

ダッシュボードはセンスも必要　　~~僕は(たぶん)センスがない…~~

Metahack/PARARSと一緒に構築！！ → たくさん救われた。

- 実は裏で色つけやらhtmlパネルやらレイアウトの修正やら、
  色々検討して確認して修正している(それに付き合ってくださっている)
- 沢山の失敗もノウハウが溜まる！
- 僕が作ったものも、提案→改善で良くなっている(設定もだけど)

色々な指摘やアイデア・検討がないと情報の可視化と利便性のある公開ページは作れない。
しかし出来ることや制限などを理解していないと、1つのグラフを作るのに高コストになることも。



今後のサービス監視

FY2021追加

Alert Mail(or other)

・通知する障害
情報を伝え

・障害の詳細
過去の障害

・ユーザにア

・運用者向け

・自動電話連

・新しく構築

・ユーザ情報やグループ情報を一見できるよう
にしたい(ex: 利用量やファイル数,逼迫状況)

・提供情報を分類してトップページを軽量化
一覧性を良くしたい

・文字を読まなくても理解できるようにしたい

・詳細情報を充実させたい

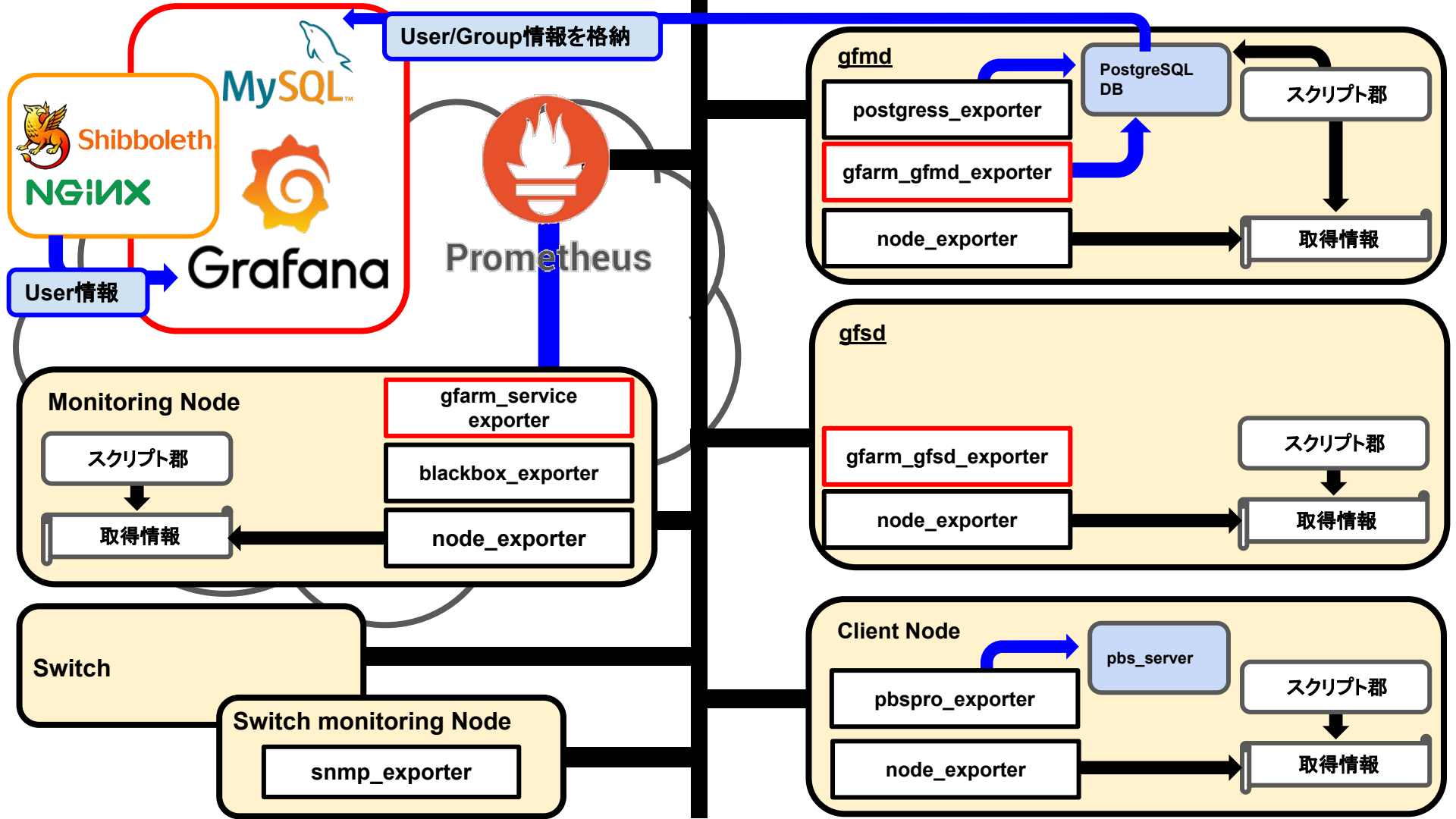
・I/O情報やサービスの遅延などもアナウンス
したい

運用情報と連携したい

・障害内容
・障害対処方法など

・稼働率・状況
・東西別情報
・ログインノード運用状況
・ネットワーク帯域など

ユーザ別情報
・利用量
・利用頻度など


User/Group情報を格納
MySQL
Shibboleth
NGINX
Grafana
User情報
Prometheus
gfmd
postgress_exporter
gfarm_gfmd_exporter
node_exporter
PostgreSQL DB
スクリプト郡
取得情報
gfsd
gfarm_gfsd_exporter
node_exporter
スクリプト郡
取得情報
Monitoring Node
スクリプト郡
取得情報
gfarm_service exporter
blackbox_exporter
node_exporter
Switch
Switch monitoring Node
snmp_exporter
Client Node
pbspro_exporter
node_exporter
pbs_server
スクリプト郡
取得情報

Gfarm Exporter

現在の方法では、メンテナンスが大変 (スクリプトの入れ替え /設置場所・結果ファイルの管理などなど)
Gfarmの運用者に向けたExporterを用意して管理したい！！

gfmdから欲しい情報
・メタデータDBの負荷(別のExporterで取得？)
・master/slaveの情報
・gfmd(メタDB)からしか取得できない情報
・保存されているデータの統計情報
・利用量　(etc…)

gfsdから欲しい情報
・I/O情報
・接続数・アクセス数 (etc…)

Client(Service Monitoring)から欲しい情報
・サービスへのアクセス状況・レスポンスタイム
・サービスの稼働情報
・接続プロセス数・ユーザ数
・(サービスの)稼働状況
・利用量(etc…)

今年度用意したスクリプトやダッシュボードを
・Exporter化
・汎用化(テンプレート化)
・運用向けの情報取得を追加
して公開したい

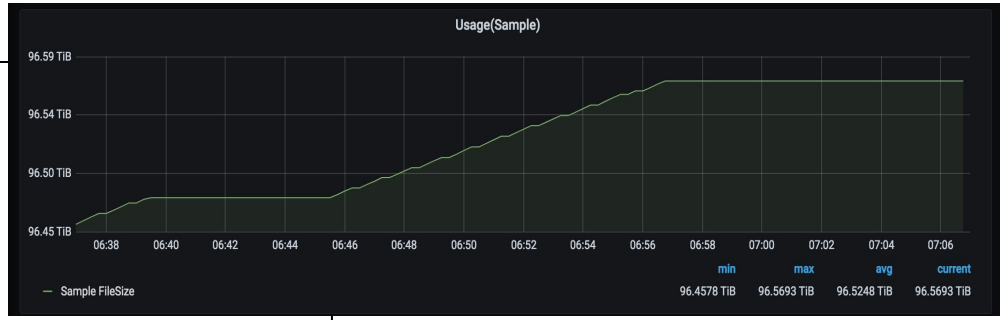
Memo: ここにGfarmのロゴを貼りたい

予約 Gfarm Exporter

例: Gfarm Service Exporter (gfusage取得)

例: gfusage -gコマンド実行結果を20秒ごとに。

```
import subprocess, sys,time,prometheus_client
from prometheus_client import start_http_server

UPDATE_COUNT = 20
GFUSAGE = prometheus_client.Gauge('get_gfusage_test',
'gfusage user size',["kind","group"])
def get_gfusage():
    result = subprocess.check_output("gfusage -g | sed 1d", shell=True)
    result_split = result.decode().strip().split('\n')

    for line in result_split:
        try:
            n = line.split(':')
            m = n[1].split()
        except:
            continue

        GFUSAGE.labels("FileSpace",n[0].replace(' ', '')).set(m[0])
        GFUSAGE.labels("FileNum",n[0].replace(' ', '')).set(m[1])
        GFUSAGE.labels("PhySpace",n[0].replace(' ', '')).set(m[2])
        GFUSAGE.labels("PhyNum",n[0].replace(' ', '')).set(m[3])

if __name__ == '__main__':
    start_http_server(9400)
    while True:
        get_gfusage()
        time.sleep(UPDATE_COUNT)
```



結果抜粋:

```
get_gfusage_test{group="<group>",kind="FileSpace"}     7.6198760053296e+013
get_gfusage_test{group="<group>",kind="FileNum"}       3.944448e+06
get_gfusage_test{group="<group>",kind="PhySpace"}      3.75988341719905e+014
get_gfusage_test{group="<group>",kind="PhyNum"}         1.7650902e+07
```

prometheus_client(python): https://github.com/prometheus/client_python

例: Gfarm GFSD Exporter (IOStat取得例)

gfsd,gfmdはI/O Statを取得している(Ganglia向けの機能として提供)
こちらをPrometheusで取得するExporterのサンプル
→ この情報はユーザに提供したい(本当はreplicationとuser I/Oが切り分けられると嬉しい)

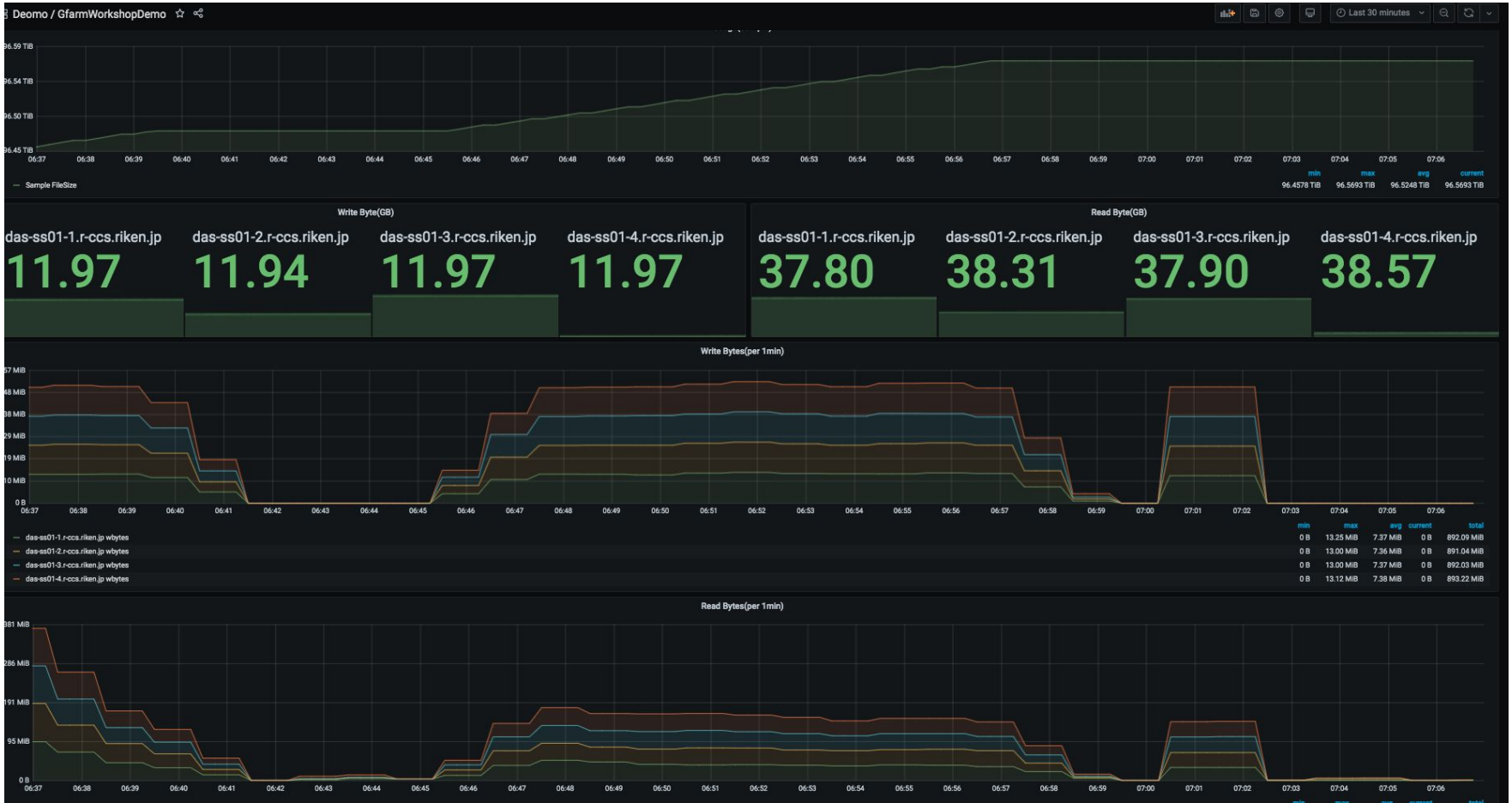
```
import subprocess,sys,time,struct,prometheus_client
from prometheus_client import start_http_server
import numpy as np

UPDATE_COUNT = 60
GFIOSTAT = prometheus_client.Gauge('get_gfiostat_test', 'get gfiostat metrics',
['kind', 'gfsd', 'cal'])

def get_gfiostat_test(hostname,iostatpath):
    f=open(iostatpath, 'rb')
    s_magic=f.read(4)
    s_nitem=struct.unpack('I', f.read(4))[0]
    s_row=struct.unpack('I', f.read(4))[0]
    s_rowcur=struct.unpack('I', f.read(4))[0]
    s_rowmax=struct.unpack('I', f.read(4))[0]
    s_item_size=struct.unpack('I', f.read(4))[0]
    s_ncolumn=struct.unpack('I', f.read(4))[0]
    s_dummy=struct.unpack('I', f.read(4))[0]
    s_start_sec=struct.unpack('Q', f.read(8))[0]
    s_update_sec=struct.unpack('Q', f.read(8))[0]
    s_item_off=struct.unpack('Q', f.read(8))[0]
    s_name=f.read(32).decode('utf-8','ignore').decode('utf-8','ignore').replace('\0','')
    item_name = []
    for i in range(s_nitem):
        s_item_name=f.read(31).decode('utf-8','ignore').replace('\0','')
        item_name.append(s_item_name)
        s_type=struct.unpack('c', f.read(1))

    dummy_read=s_item_off - (56 + 32 + (s_nitem * 32))
    f.read(dummy_read)
```

```
# item value
    all_item_value = []
    for i in range(s_rowmax):
        s_valid=struct.unpack('Q', f.read(8))[0]

        item_value = []
        for j in range(s_nitem):
            s_vals=struct.unpack('Q', f.read(8))[0]
            item_value.append(s_vals)

        all_item_value.append(item_value)
        # gap
        f.read(s_item_size - 8 - (s_nitem * 8))

    total=np.sum(all_item_value, axis=0)

    GFIOSTAT.labels(item_name[0],hostname,"total").set(total[0])
    GFIOSTAT.labels(item_name[1],hostname,"total").set(total[1])
    GFIOSTAT.labels(item_name[2],hostname,"total").set(total[2])
    GFIOSTAT.labe

    f.close()


if __name__ == '_
    start_http_serv
    while True:
        (snip)
```

結果例:

```
get_gfiostat_test{cal="total", gfsd="das-ss01-1.r-ccs.riken.jp", kind="wbytes"} 24670569072851
get_gfiostat_test{cal="total", gfsd="das-ss01-2.r-ccs.riken.jp", kind="wbytes"} 23478128446961
get_gfiostat_test{cal="total", gfsd="das-ss01-3.r-ccs.riken.jp", kind="wbytes"} 24992743276385
get_gfiostat_test{cal="total", gfsd="das-ss01-4.r-ccs.riken.jp", kind="wbytes"} 21657095295711
```

DEMO_2: https://hpci-web01.r-ccs.riken.jp/grafana/d/GFARM_WORKSHOP0304DEMO/gfarmworkshopdemo?orgId=2



まとめ

- HPCI共用ストレージサービスの稼働情報のユーザ公開を開始した
- 監視ツールはPrometheus / ダッシュボードツールはGrafanaを活用
- Gfarmのコマンド等を駆使して情報を可視化した
- 以下を強化したい!!
  - ユーザ別・グループ別情報
  - 改善、情報強化を予定
  - Gfarm向けのExporterを開発予定
- Gfarmへの要望
  - 監視サーバに引き渡しやすいフォーマットで情報が取得できると嬉しい
    - 例えば、gfstatus -Mのようなフォーマット
    - 例えばJSON(原田さん案流用!!)
    - コマンドごとにPaserを実装しないといけないので(使い回しができない)
  - それはそれとしてユーザコマンドで必要な情報が取得できるので大変助かりました!!
- 監視サービスの名前を募集します！！

  (例: pig nose / sky-eye / Hs5[HPCI.shared.storage.services.stats.service]....)

ぼつ

例: サービス稼働率の取得

追記・修正

情報取得手順
(1) モニタリング用のクライアントを準備
(2) 稼働状況・稼働率を定義
(3) 情報取得の仕様を準備
(4) Collection向けのスクリプトを準備
(5) node_exporter配下に情報取得ファイルが作成されるように設定
(6) node_exporter -collector.textfile.directoryで情報取得設置ディレクトリを指定
(7) Prometheusで情報取得確認
(8) Grafanaでダッシュボード準備

gfarm_operation_stat{ instance="ss-service-monit-client01:9100",<snip>} **1 (1:正常/0:停止)**

eBPF/コンテナ監視