仕様追加

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | プログラム |
| 2015/11/19 | 10章追加 | Wロック時の障害対応 | 修正済み |
| 2015/11/27 |  | セッションクローズ時にエフェメラルファイルのコピーファイル | 修正済み |
| 2015/12/27 |  | デレゲーション所有者指定APIを用意した | 修正済み |
| 2015/12/31 |  | イベントにイベント通版を導入した | 修正済み |
| 2016/1/20 |  | QueuSuspend/Resume/Maxの追加 | 修正済み |
| 2016/1/26 |  | Page　overflowの抑止 | 修正済み |
|  |  |  |  |

# イベントループ

イベント制御については、Paxos\_2ではepollによる方式に変更する。

理由は、tgtd等で一般的となっているからである。また、epollではselectと異なり、横からイベントの登録削除ができるからである。

　なお、さらにlibevent対応とした。（2015年9月）これにより、epoll、selectは隠蔽化され、またWindows　SDKにも対応できることとなった。（NWGadget.cは対応済み）

## FdEventCtrlCreate /FdEventCtrlDestroy

　EVENT\_MAX（現在は1000）のepollの作成/削除とハッシュ、リストの初期化/破壊を行う。

## FdEventInit

Int　FdEventInit( FdEvent\_t \*pEv, int Type, int Fd, int Events,

void \*pArg, FdEventHandler\_t fHandler )

　FdEvent\_t構造体に情報を設定する。

　Type AP用

　Fd ファイルディスクリプタ

　Events EPOLLIN,EPOLLOUT等

　pArg ハンドラーへの引数

　fHandler ハンドラー

## FdEventAdd

Int　FdEventAdd( FdEventCtrl\_t \*pFdEvCt, uint64\_t Key64, FdEvent\_t \*pEv )

　FdEventを登録する。Key64はハッシュ登録のキー。

　Key64は、一般にはファイルディスクリプタだが、本件では接続切断を超えてリソースを維持しなければならないセッション情報（Accept\_t）があるのでキーは別途用意されるとしている。

## FdEventDel

Int　FdEventDel(FdEvent\_t \*pEv )

　FdEventを削除する。

## FdEventSuspend

　全てのFdEventをサスペンドする。

ロック保持でハンドラーが呼び出される。

## FdEventResume

Int　FdEventResume( FdEventCtrl\_t \*pFdEvCt, FdEvent\_t \*pEv1 )

　pEv1をレジュームする。pEv1がNULLであればすべてをレジュームする。

ロック保持でハンドラーが呼び出される。

## FdEvCtLock/FdEvCtUnlock

　FdEventハンドリングの排他制御をする。

## FdEventLoop

　イベント時にハンドラーを呼び出す。

## ハンドラー

Int　Handler( FdEvent\_t \*pEv, FdEvMode\_t　Mode )

ModeはEV\_LOOP、EV\_SUSPEND、EV\_RESUMEも飛び出しモードである。

EV\_SUSPEND/EV\_RESUMEはエラー発生時に呼び出される。

# 管理ポートによる管理コマンド

　AF\_UNIXソケットで定義する。

　管理コマンドはlocalに投入し、リモートからはssh等でsecurityを別途確保する。

## Paxos管理ポート

　「/tmp/PAXOS\_ADMIN\_PORT\_%cell\_%Id」とする。

セル名を指定するのは複数セルが同一マシンに載せられるからである。

本ポートはpaxosモジュールのFdEventループに登録される。

## Session管理ポート

　「/tmp/PAXOS\_SESSION\_ADMIN\_PORT\_%cell\_%Id」とする。

セル名を指定するのは複数セルが同一マシンに載せられるからである。

本ポートはsessionモジュールのFdEventループに登録される。

## AP管理ポート

　通常はsession管理ポートを使用するが、session管理ポートはsession通信プロトコル下の制約があるため、AP管理コマンドをこの制約下にするのが困難である場合にAP管理ポートを用意する。

　例えば、cmdbでは、「/tmp/PAXOS\_CMDB\_ADMIN\_PORT\_%cell\_%Id」とする。

# 動的メンバー変更

　PaxosCellは、構成ファイルを変更し、Idで立ち上げると、ハートビートで他サーバーに自アドレスが通知され、自動的に変更される。

　SessionCellは、PaxosSessionChangeMemberコマンドによりセルに通知され、接続中のクライアントに自動的に通知される。新たに起動されるクライアントは構成ファイルを参照するので、構成ファイルは予めクライアントノードに配備されていなければならない。

## 構成ファイル

　構成ファイルを変更し配備する。

## PxaosCell

　Paxosレベルでは、変更対象サーバはダウンし新たなサーバが参入することを想定する。

　生きているサーバはSendAliveでハートビートを送っている。受信時、アドレスが異なれば更新する。これにより自動的にメンバー変更が行われる。

　cold　replace

　　ノードが障害等で離脱し、新たなノードをスクラッチから再参入させる。

　warm　replace

　　動作中のノードを新たなノードに置き換える活性保守を想定する。

新たなノードをエクステンションで同期させホット状態とする。新たなノードをダウンし、次に置き換え対象ノードをダウンする。新たなノードを対象ノードのIdで再参入する。

この時の信頼性を考察する。新旧ノードのダウン、新ノードの再参入に要する切り替え時間を⊿tとする。1個あたりのMTTFをT、現在のコア稼動数をnとすると、MTTF(n)=T/nがセルの平均故障時間となる。コア数が減れば平均故障時間が増える。旧ノードをダウンさせた時、MTTF(n-1)=T/(n-1)であり、これに対する故障率を⊿t/T\*(n-1)と考えることができる。すなわち、切り替え時間がMTTFより充分小さければサービスが継続されるとしてよい。

なお、たとえ故障して過半数を維持できなくなりサービスが中断したとしても、生きているサーバーが居れば、参入サーバーで過半数が回復され、速やかにサービスが継続される。

## SessionCell

　管理コマンドはセルに対するコマンドとして投入することで同期的にする。これにより、参入サーバも変更ができる。

SessionCellには版を導入し、サーバはクライアントに常に版を通知する。クライアントは版が異なればSessionCellをサーバから取得する。

コマンド

「PaxosSessionChangeMember　cell　id　in\_addr　port」

Cell 対象セル

Id 対象id

In\_addr 新しいIPアドレス

Port 新しいポート番号

説明

　対象セルに対し、対象idのIPアドレスとポート番号を指示する。これはPaxos合意で処理される。

　接続中のクライアントには変更が通知される。

# サーバ構成数のパラメータ化

## サーバ

　サーバ起動時に構成数を指定できるようにする。

## クライアント

　構成ファイルのレコード数でよい。

# 動的RAS追加

　Rasセルを起動する前の既に起動されたアプリケーションがRasセルに登録する方式を検討する。

　アプリケーションがセルサーバである場合は、管理ポートからRASセルを通知し登録せしめる。

　PFSAdmin　cell　Id　ras　RASセル名

とする。ローカル上のAF\_UNIXポート(PAXOS\_SESSION\_ADMIN\_PORT\_%cell\_%Id)から通知され、サーバはRASセルに登録及びマスタ変更を瞬時に行うための監視スレッドを起動する。（FdEventループに登録しない）

## PFSRasThreadCreate

PaxosSessionAllocでpSessionを獲得する。

PFSRasThreadCreate(pAcceptor,pSession)は、

1. PaxosSessionOpenによるRASセル接続
2. PFSRasRegisterによる登録
3. スレッド作成
4. 自セルの他メンバの切断イベントによるマスター選び

を行う。

　異常時にはスレッドは後始末を行い消滅する。

# 資源

## CPU負荷

　クライアント接続数で代替する。

## メモリ量

　それぞれのオブジェクト型毎に自分で管理する。あるいは、getrlimitから推定する。

# エラーコードとログ

## ログ

　(プロセス　3>&1　1>&2　2>&3　|　LogPrint　)　|&　syslog

とするとNWGagdetのログがstderrからstdoutに変更される。

LogPrintの出力と元のstdoutの出力がsyslogに出力される。

# エラーチェック（ソフトで実行）

　システム要素が３要素より構成されるとすると、

PU（CPU） ECCによる

SU（Storage） BCCによる

CU（通信） TCP/UDP/IPはチェックサムによる

が一般的である。

本システムでは、上記チェック及び回復機能がないとして、ソフトウェアで実装することを考える。その際、サーバシステムは受動的であり、クライアントからの投入（刺激）でサーバが変化することに鑑み、チェックサムをクライアントで計算し、サーバで持ち回りしようすることにすれば効率的である。

また、サーバでは差分のみの計算とできれば秀逸である。これには、1の補数（64ビット）によるチェックサムが最適である。なお、本システムでは1ビットエラーを検知する[[1]](#footnote-1)だけでよいので1の補数で充分である。さらに、計算は1回走査するだけなので速い。

チェックエラー時にはシステムをアボートする。

## 制御データと本体データ

基本的に本体データについてチェックサムを行う。

制御データ（リスト等）にエラーが生じればアボートとなることを想定する。

HASHについては、照合したときにハッシュ値を再計算する。1ビット異なればハッシュ値が異なる。

## 1（64ビット）の補数和のアルゴリズム

1の補数和は、2進数の和でMSBからの桁上がりをLSBに足しこむ、ことで得られる。

A+B=Cで桁上がりはCarry=((A,B)>C)で判断できる。桁上がりが発生したら、LSBに足しこむ(C+=Carry)。

　補数和の差は、否定(~A)を桁上がり付で足しこむ、ことで得られる。これにより、旧部分を引き、新部分を足すことで新しい補数和を得ることができる。

なお、補数和は先頭から8バイト境界で計算するので、新旧の差分計算時には境界内の何番目かに注意しなければならない。

　また、Endianの問題がある。これについては、データをchar　c[8]にコピーし、w64=\*(uint64\_t\*)cで解決する。（境界を考慮して、cはuint64\_t　c64と宣言する。）

## プロトコルヘッダ（通信）

プロトコルヘッダのh\_Magicをh\_Cksum64に変更し送信時に計算(h\_Cksum64=0)設定する。サーバでは受信時にはチェックをせず、使用時にチェックする。クライアント側ではAPがチェックするか不明なのでチェックサムが0以外であればチェックをする。（1の補数和の否定の場合、チェックサム計算を行うと0以外となる。

## キャッシュとファイル（HD）

HD用のキャッシュはブロック単位（64KiB）であるが、少量データアクセスの度にチェックするのではコストが大となりすぎる。メモリ上ではサブブロック（4KiB）ごとにチェックサム計算をする。

ファイルをHDに永続化する場合には、ブロック単位のチェックサムを別ファイルとする。これにより、S3アプリ等でローカルに直接データにアクセスすることができる。HDからのロード時にチェックする。

なお、前回チェックより一定時間（msecオーダー？）が経っている場合にもブロックごとのサブブロックのチェックを行う。

## 帯域外データ

　クライアントからチェックサム付でサーバに送信される。チェックはvalid時に行う。

データ部のメモリ容量を管理しオーバー(100MiB?)すればHDにセーブ／ロードを行うが、ロード時にのみチェックする。

　他サーバからの要求時には、差分のみを設定し直す。

## キューデータ

　データ総体について送信時にチェックサム設定、受信時にチェックを行う。

## ロック

　データサイズが小さいので行わない。HASHでチェックされる。

# ワーカー方式

　接続毎のスレッドからepoll方式に変更したので、並列性を確保するためにワーカースレッド方式を採用する。

Listenポート

Acceptﾊﾝﾄﾞﾗｰ

管理ポート

通常ポート

Recvﾊﾝﾄﾞﾗｰ

Worker

## 受付処理ワーカー

　通常処理のワーカーで複数起動される。

★FdEventハンドラーは、ワークキューに受信データを次から次へとキューイングする。フロー制御は上位アプリケーションで行われるとし、キューリストの長さによるフロー制御は行っていない。

## 実行処理ワーカー

　合意実行は、現在単一スレッドである。分割処理ができるようであれば振り分けをおこなうことも考えられる。

## フリーズスレッド

　リモートからのフリーズ要求については、自分以外の受付要求イベントの入り閉塞を行う。これにより、他のクライアントからの要求はサスペンドされる。

　これに専用スレッドを割り当てておく。

# デレゲーション対応

## 障害検知時の当該ファイルの削除

　デレゲーションは、PFSLockW/Rを使用する。

Lockにはデータを付与していないので、ロックに対応するデータが必要な場合には、ロック名のファイルを想定する。クライアントがWロック中に障害でダウンした場合には無条件にこのファイルを削除する。

## 所有者指定API

　スレッド以外の所有者を指定できるAPIを用意した。

# エフェメラルファイル対応

## セッションクローズ/アボート時のバックアップファイル作成

　セッションクローズ・アボート時にエフェメラルファイルが削除されデータが失われる。

これを削除する前にファイル名「エフェメラル名\_BAK」でコピーするようにする。

これにより、監視モニターは障害直前のデータを取得することができる。

## Path\_tにエフェメラルフラグ

# イベント対応

## イベント通版

　重複チェック用に、イベント通番を、ディレクトリ、ロック、キューおよび通信プロトコルに導入した。

# Queue対応

## QueueSuspend/QueueResume/QueueMax

　入り閉塞時、メッセージを廃棄していたが、保留することにした。

このため、QueueSuspend/QueueResumeを導入し、列に上限を設けるためにQueueMaxを用意した。

　これにより、再参入のフリーズ時のメッセージが保留され、イベントがスムーズに継続することになる。

# PaxosのPageOverflow抑止

　各Paxosは他のサーバ用に旧ページを保持している。Majorityがどんどん進みminorityがなかなか追いつかないとメモリが大量に消費されてしまう。

1. PutPageで旧ページ保持が一定数以上であれば抑止フラグをonとする。
2. RecvBeginではonのときSupressedとし、応答を返さない。

これにより、現在ページで進行が止まる。

1. MinorityはCatchupで追いつく。
2. Heartbeat受信でページが追いついたところで、抑止フラグをoffとし、Release\_suppressed\_acceptでacceptを実行させる。

1. 1ビットエラー障害のみを考える。瞬時（単位時間内）の多重故障は物理学的には対象外である。 [↑](#footnote-ref-1)