ANCL

ShareTask 5.5

(doc#05)

サーバー

運用管理マニュアル

アンクル
2014 **年** 2 **月**

目次

図目次

表目次

1 はじめに

この文書は, ShareTask によるクラスターの管理手順について説明したもので, クラスターの管理者を対象にしています.

2 ログイン/ログアウト

ShareTask にログインするには, ShareTask サーバーの URL にブラウザでアクセスします.

図 2.0.1 ShareTask ログイン画面

ID には, 認証システム (LDAP 等) に登録されているユーザー名を, パスワードには該当するパスワードをタイプしてください.

ShareTask からログアウトするには, 画面の右上端にある [LOGOUT] をクリックしてください.

3　管理者を限定する

ShareTask の管理者は, root ではなく, ShareTask にあらかじめ管理者として登録されているユーザーです. ShareTask をインストールした初期状態では, ログインできるすべてのユーザーが管理者権限を持っていますが, 「システム設定」画面のパラメーター群の冒頭にある

1 **管理者** UID

の値をゼロから変更することによって, 限定されたユーザーのみが管理者権限をもつことになります. たとえば, 501 とすれば, UID=501 のユーザーのみが管理者権限をもちます. UID は空白文字で区切って複数列挙できます.

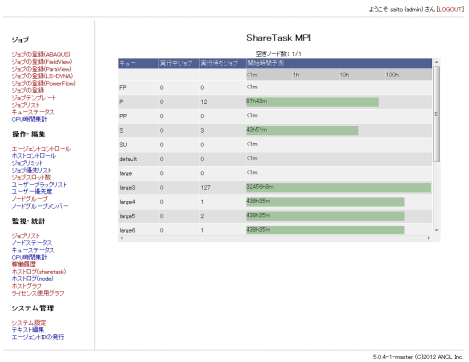

図 3.0.1　管理者画面

図 3.0.2　システム設定画面

4 プロセス

4.1 ShareTask サーバーで動いているプロセス

ShareTask サーバー上では, 以下のプロセスが常駐しています.

- `Apache`
- `PostgreSQL`
- `memcached`
- `sharetaskd`

4.2 計算ノードで動いているプロセス

計算ノード上では, 以下のプロセスが常駐しています.

- `java`

5 設定

ShareTask の設定は, ShareTask サーバー上の設定と, 計算ノード上の ShareTask エージェントの設定に分かれています.

アプリケーション (バージョン) を追加する場合は, ShareTask サーバーとエージェントの両方の設定変更が必要です.

設定ファイルの関係を図 5.0.1 に示します.

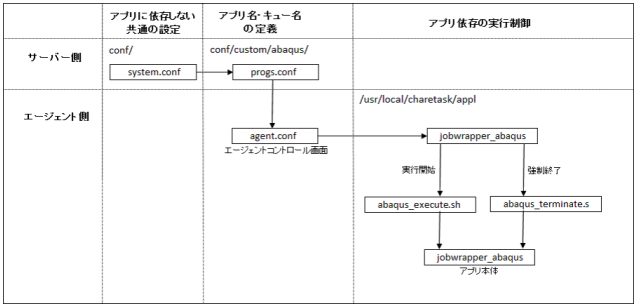

図 5.0.1　設定ファイル関係図

ShareTask の基本的な設定は, ShareTask サーバーのインストールディレクトリの `conf` ディレクトリにある `system.conf` ファイルを編集することによって行います.

`/home/sharetask/sharetask_server/conf/system.conf`

5.1 基本設定

ShareTask の基本設定は, `conf` ディレクトリに配置された `system.conf` を編集することで行います. 本節では, この `system.conf` の設定パラメーターについて説明します.

DISPAT

エージェントが起動する, アプリケーションプログラムを束ねるラッパースクリプト (ディスパッチャーと呼んでいる) の名称を指定します. ディスパッチャーは, 与えられた引数から起動すべきアプリケーションを決定し, その実行ファイルを起動するように記述します. ディ

スパッチャーの使用により, エージェント設定ファイルの `define_program` セクションを集約して記述できますので, 記述量を圧倒的に減らせます.

例 :

```
define_program {
   name = dispat
   executable = /usr/local/sharetask/appl/dispat.sh
   ...
```

このパラメーターを指定した場合, 従来のプログラム名の代わりに, ディスパッチャーの第 1 引数をアプリケーション識別子とみなし, ジョブのリソース使用量集計などにおいてキーとして扱います.

CACHE
待ち時間予測の算出を高速化するために, `memcached` を使用するかどうかを選択します.

PROGS_SELECTOR
ジョブ登録画面で, プログラム名とキューをプルダウンメニュー選択方式に切り替えられます. 値 1 でプルダウンメニュー選択方式を指定, 値 0 で従来の任意文字列タイプイン方式を指定します. プルダウンメニューの選択肢は, `conf/progs.conf` に記述します.

NO_UPLOAD
このパラメーターの値を 1 に設定すると, ジョブ登録画面の入力ファイル指定が, 実行ノード側のファイルシステムのパスを指定するものであるとみなします. つまり, ローカルファイルのアップロードはしないことになります. 値 0 では, 入力ファイルはブラウザが動いているローカル上のパスであるとみなし, アップロードが行われます.

JOBLIST_SHOW_ALL
一般ユーザーが開くジョブリスト画面に, 全ユーザーのジョブを表示するかどうかを指定します.

値	動作モード
0	当該ログインユーザーのジョブのみを表示する.
1	全ユーザーの全ジョブを表示し, 全ジョブについてジョブ詳細画面を表示でき, 入出力ファイル等を閲覧できる.
2	全ユーザーのジョブを表示するが, ジョブ詳細画面を表示できるのは当該ログインユーザー本人のジョブのみに限定される. 他ユーザーのジョブの存在を知ることはできるが, その詳細は閲覧できない.

AGENT_AUTO_REG

エージェントからサーバーへの初回アクセスにおいて, サーバーが当該エージェントを自動的に登録し認証情報を発行するかどうかを指定します. 値 1 では, 新しく起動したエージェントは自動的にサーバーに登録されます. 値 0 では, 自動登録を許可しないので, 管理者用メニューの「エージェント ID の発行」を使用して認証情報ファイルを取得し, これを手動でエージェント側の`/var/spool/sharetask/auth` ディレクトリに配置する必要があります.

AUTH

ユーザー認証方式を選択します. 指定できる値は, 次の 3 種類です.

`sharetask`	ShareTask が内蔵するユーザーデータベース
`nis`	NIS サーバーと連携します
`ldap`	LDAP サーバーと連携します

MIN_UID

MAX_UID

NIS, LDAP の場合に, ShareTask ユーザーとして認識するユーザー ID の最小値と最大値を指定します. この範囲外のユーザー ID のユーザーは ShareTask ユーザーとして見なされず, ShareTask にログインできません.

NIS_DOMAIN

上記 AUTH で `nis` を指定した場合に有効なパラメーターで, NIS のドメイン名を指定します.

NIS_MAP

上記 AUTH で `nis` を指定した場合に有効なパラメーターで, NIS のユーザー認証のマップ名を指定します. 一般的には, `passwd.byname` です.

以下は, 上記 AUTH で `ldap` を指定した場合に有効なパラメーターです.

LDAP_TYPE
LDAP サーバーに管理者権限で問い合わせるかどうかを指定します.

LDAP_HOST
LDAP サーバーのホスト名あるいは IP アドレスを指定します.

LDAP_SSL
LDAP_TLS
LDAP サーバーとの通信の暗号化方式を選択します.

LDAP_SASL
SASL(Simple Authentication and Security Layer) を使用するかどうかを指定します.

LDAP_ATTR_ID
ユーザー ID が納められている属性名を指定します.

LDAP_ATTR_PASSWORD
パスワード文字列が納められている属性名を指定します.

以下は, 上記 LDAP_TYPE=admin で管理者権限での問い合わせを指定した場合に有効なパラメーターです.

LDAP_ADMIN_DN
管理者のドメイン情報 (`cn, dc`) を指定します.

例： `LDAP_ADMIN_DN = cn=Manager,dc=anclab,dc=com`

LDAP_ADMIN_PASSWORD
管理者のパスワードを指定します.

LDAP_BASE_DN
ベースとなるドメインを指定します.

例： `LDAP_BASE_DN = dc=anclab,dc=com`

LDAP_FILTER
ShareTask ユーザーとして許可する範囲を抽出する絞り込み条件を指定します (書式については RFC 2254 を参照).

例： `LDAP_FILTER = (|(l=Tokyo)(l=Osaka))`

5.2 ジョブパラメーターフォームの設定

ジョブテンプレート画面のジョブパラメーターフォームのプルダウンメニューを変更するには, conf ディレクトリにある progs.conf を編集します. 本節では, progs.conf の記述形式について説明します.

progs.conf の内容は, あるアプリケーションに着目した場合の, キュー, CPU 数の組み合わせを 1 行で記述するものです.

形式

```
appl => queue1[ncpu, ...], queue2[ncpu, ...]
```

例：

```
abaqus => para02[2], para04[4], para08[4,8], para16[8,16]
```

この記述例が意味するところは以下の通りです.

- abaqus というアプリケーションを選択すると, キューのプルダウンメニューの選択肢は, para02，para04，para08，para16 の 4 種類である.
- para02 を選択すると, CPU 数のプルダウンメニューの選択肢は 2 だけである.
- 同様に, para04 を選択すると, CPU 数のプルダウンメニューの選択肢は 4 だけである.
- para08 を選択すると, CPU 数のプルダウンメニューの選択肢は 4，8 の 2 種類である.
- para16 を選択すると, CPU 数のプルダウンメニューの選択肢は 8，16 の 2 種類である.

CPU 数の指定には, 次のオプション指定が可能です.

形式：

```
appl => queue[n1-n2]
```

$n1, n2$ は 1 以上の整数

$n1 \leq x \leq n2$ のすべての整数値を選択肢とします.
プルダウンメニューの初期値は n1 となります.

形式：

`appl => queue[n1-n2:n3]`　　$n1, n2$ は 1 以上の整数, $n1 \leq n3 \leq n2$

$n1 \leq x \leq n2$ のすべての整数値を選択肢とします.
プルダウンメニューの初期値は `n3` となります.

形式：
`appl => queue[{nodes} × {cpus}]`　　`nodes, cores` は 1 以上の整数

1 ノードあたり `cpus` 個の CPU 数で, `nodes` 台のノードを割り当てます. 全体で `{nodes}`×`{cpus}`個の CPU を割り当てます.

5.3 キューへのアクセス権設定

あるキューを特定のユーザーにだけに使用させたい場合が考えられます. この設定のために, キューアクセスコントロールファイル

```
$SERVER_HOME/conf/acl/queue.acl
$SERVER_HOME は, 通常は /home/sharetask/sharetask_server
```

があります. このファイルは, 次の表 5.3.1 の記法で, キューに対するユーザーのアクセス権限を記述します.

表 5.3.1　アクセス権限の記述

	記述行パターン	意味
1	+:QUEUE_REGEXP:GROUP_REGEXP	マッチするユーザーグループについてキューへのアクセスを許可する
2	-:QUEUE_REGEXP:GROUP_REGEXP	マッチするユーザーグループについてキューへのアクセスを許可しない
3	+g:QUEUE_REGEXP:GROUP_REGEXP	1 と同義
4	-g:QUEUE_REGEXP:GROUP_REGEXP	2 と同義
5	+u:QUEUE_REGEXP:USER_REGEXP	マッチするユーザーについてキューへのアクセスを許可する
6	-u:QUEUE_REGEXP:USER_REGEXP	マッチするユーザーについてキューへのアクセスを許可しない

`QUEUE_NAME_REGEXP` は, キュー名の正規表現

キュー名は, エージェント設定ファイルの `define_program{version}`で定義された名前です.

`USER_NAME_REGEXP` は, 行頭が, `+u:`あるいは`-u:`の場合はユーザー名, `+: +g: -: -g:`のいずれかの場合は, ユーザーグループ名の正規表現として解釈されます.
ユーザー名は, PAM, NIS, LDAP をサポートしています.
ユーザーグループ名は, PAM, NIS をサポートしています.

6 キューの登録・削除

キューの登録・削除は, progs.conf と, エージェント設定ファイルの編集によって行います.

progs.conf の編集は, 5.2 節で説明しました. エージェント設定ファイルは, エージェントコントロール画面で行います. エージェントコントロール画面では, 各計算ノード毎にエージェント設定ファイルを割り当てられます. エージェント設定ファイルでは, 当該計算ノードが担当するアプリケーションとキューの組み合わせを記述します.

エージェント設定ファイルに記述したアプリケーションとキューの組み合わせが, progs.conf で設定したアプリケーションとキューの組み合わせと整合していなければなりません.

6.1 エージェント設定ファイルの例

```
define_resource {
   max_procs = 8
   polling_interval = 60
   keepalive_interval = 15
}
define_program {
   name = abaqus
   version = para02
   wrapper = /usr/local/sharetask/appl/jobwrapper_abaqus.sh
   executable = /usr/local/sharetask/appl/abaqus_execute.sh
   workdir_parent = /var/spool/sharetask/jobsv
}
```

この設定例では, 以下を記述しています.

max_procs = 8

このエージェント設定ファイルが反映される計算ノードでは, 8 個までのジョブプロセス (MPI ではランクプロセス) を賄えます. 計算ノードが持っている CPU 数とも解釈できますが, 物理的な CPU 数と一致していなくてもかまいません. たとえば, 物理的には CPU が 8 個あっても, `max_procs = 4` と記述すれば, 4 プロセスまでしか実行しません. 逆に `max_procs = 16` と記述すると, 物理的には CPU が 8 個しかないのに 16 プロセスまで実行してしまいます.

polling_interval = 60

`max_procs` で設定されたプロセスに達していない状況, すなわち CPU に空きがある状況において, ShareTask サーバーに対して新たなジョブの問い合わせを行う時間間隔を 60 秒に設定します.

keepalive_interval = 15

ジョブを実行開始した状況において, 当該ジョブの死活監視報告を ShareTask サーバーに対して送信する時間間隔を 15 秒に設定します. これが一定時間以上途絶すると, 当該ジョブ実行が障害によって失われたと判定され, 1 時間後にはキュー上で待機状態に戻されます. これによって, 障害によって実行中に失われたジョブが自動的に再実行されます.

name = abaqus
version = para02

`abaqus` というアプリケーションで, `para02` というキューに投入されているジョブについて, ShareTask サーバーに問い合わせます. 問い合わせの時間間隔は, `polling_interval = 60` に従います.

wrapper = /usr/local/sharetask/appl/jobwrapper_abaqus.sh

ShareTask サーバーにジョブを問い合わせた結果, 該当するジョブを払い出された場合, 当該ジョブを実行するためのラッパースクリプト (一般的には Bash シェルスクリプト) が `/usr/local/sharetask/appl/jobwrapper_abaqus.sh` であると指定します. このラッパースクリプトは, アプリケーションを起動するための前処理, 後処理を行います. 起動すべきアプリケーションは, 次の `executable` で指定します.

executable = /usr/local/sharetask/appl/abaqus_execute.sh

起動するアプリケーションの実行形式ファイルとして,
`/usr/local/sharetask/appl/abaqus_execute.sh` というシェルスクリプトを指定します. `a.out` を直接指定してもよいのですが, シェルスクリプトでラッピングして柔軟性を高

めています.

workdir_parent = /var/spool/sharetask/jobs
ジョブの実行ディレクトリとしてテンポラリーディレクトリを作成しますが, そのベースディレクトリを/var/spool/sharetask/jobs と指定します. テンポラリーディレクトリは, このサブディレクトリとしてジョブ ID を名前として作成されます.
例： /var/spool/sharetask/jobs/12345
このテンポラリーディレクトリに, **実行開始に必要なファイルがすべてコピーされます**.

6.2 エージェント設定ファイルのパラメーター

以上, 例で説明したエージェント設定ファイルのパラメーターを表 6.2.1 にまとめます.

表 6.2.1 エージェント設定ファイルのパラメーター

キーワード	意味	値
define_resource	共通パラメーターを記述するセクションの開始を宣言する	
max_procs	この計算ノードから払い出せる CPU コアの最大数	正の整数
polling_interval	ジョブ仲介サーバーにジョブを問い合わせる間隔	正の整数 (秒)
keepalive_interval	ジョブ仲介サーバーにジョブ実行状態を通知する間隔	正の整数 (秒)
group	エージェントが属するグループの ID(ノードグループ画面で定義した ID)	正の整数
rsrc_monitor	リソース情報を取得する外部プログラムのパス:標準形は次; /usr/local/sharetask/bin_agent/rsrc_monitor.sh	パス文字列
rsrc_monitor_interval	rsrc_monitor を起動する周期 (秒数)	正の整数 (秒)/ 標準値:60
report_resource_interval	エージェントが稼働しているホストの属性値と資源量の状況をサーバーに報告する間隔 (秒数)	正の整数 (秒) 標準値：60
report_rrd_interval	ホストグラフ画面のリソースデータをサーバーに報告する周期 (秒数) / 指定しなければ報告しない	正の整数 (秒) 標準値：300
rrd_source	ホストグラフ画面のリソースデータ源を指定する / agent からの報告を採用する / snmp/linux sgaretaskd から SNMP によるポーリングを採用する	agent snmp/linux

キーワード	意味	値
upload_size_limit	巨大なファイルをサーバーに分割アップロードする際の分割サイズ / サイズ指定には, 単位 (K, M, G) が使用できる	サイズ指定 1024000000 / 1G
upload_schedule	実行テンポラリディレクトリ (/var/spool/sharetask/jobs/{JOBID}) のファイルをジョブ実行中にサーバーにアップロードするスケジュール / $\%n$ 経過時間を n で割った余がゼロのときに真 / $< n$ 経過時間が n より短い / $> n$ 経過時間が n より長い / 例: %1<5, %5>5 / 5 分未満では 1 分間隔で, 5 分を超えた場合は 5 分間隔でファイルをアップロードする	スケジュール記述
snapshot_upload	分割アップロードをするかどうか	true : する / false: しない
split_upload	snapshot_upload の別名 / 将来, この名前に統一される	
debug	エージェントログファイル /var/log/sharetask-agent.log のログレベルをデバッグにする	true : する / false : しない
auto_cleanup	実行テンポラリ (/var/spool/sharetask/jobs/{JOBID}) をジョブ実行終了後に削除せずに残す	true : 残す / false : 削除
restart	エージェント (Java プロセス) を再起動するためのシーケンス ID / 前回のシーケンス ID よりも大きい値が指定されていると再起動する / はじめて記述したときは, 値の大小にかかわらずに再起動する / 再起動は, /etc/init.d/sharetask-agent restart による	正の整数

キーワード	意味	値
define_program	ジョブキューを記述するセクションの開始を宣言する	
name	アプリケーションの名前	文字列
version	ジョブキューの名前	文字列
executable	アプリケーションの実行ファイルのファイルパス	パス文字列
wrapper	ジョブ実行を監視制御するためのラッパープログラム	パス文字列
workdir_parent	ジョブ実行のテンポラリディレクトリの親ディレクトリ	パス文字列
polling_interval	ジョブ仲介サーバーにジョブを問い合わせる間隔	正の整数 (秒)
rmonitor	ジョブが使用している資源量を検出するスクリプトのパス / 標準形 : /usr/local/sharetask/bin_agent/jobmon	パス文字列
back_to_waiting	アプリライセンスが満たされなかった場合に, 実行中から待機中に遷移するモード (EXPERIMENTAL)	true : 使用する false : 使用しない (デフォルト)
define_server	ジョブ仲介サーバーへの接続情報を記述するセクション	
base_url	ジョブ仲介サーバーのベース URL	[http|https]://...stask
name	ジョブ仲介サーバーの管理上の識別名	文字列
priority	冗長構成の際の優先順位	ゼロ以上の整数
cpu_time_limit	CPU 時間の上限	秒数 (小数点使用可)
vm_size_limit	VM ピークサイズの上限	単位は MB

- executable の値が / 以外の文字から始まる場合は, ShareTask のインストールディレクトリ ($SHARETASK_HOME) からの相対パスとして解釈される.
- wrapper の値が / 以外の文字から始まる場合は, ShareTask のインストールディレクトリ ($SHARETASK_HOME) からの相対パスとして解釈される.
- wrapper として, $SHARETASK_HOME/bin_agent/jobwrapper0(シェルスクリプト) が提供されています (エージェント インストールマニュアル (doc#02)「6.4 設定ファ

イルの例 2」を参照).

```
wrapper = bin_agent/jobwrapper0
```

- `time_limit` は, `wrapper` として `bin_agent/jobwrapper0` を設定したときに有効になる.
- `wrapper = bin_agent/jobwrapper0` を設定しているにもかかわらず, `time_limit` を設定しないときは打ち切り時間は設定されず, 無限に実行される.
- `workdir_parent` の値が / 以外の文字から始まる場合は, ShareTask のワークディレクトリ (`$SHARETASK_WORK`) からの相対パスとして解釈される.
- `$SHARETASK_WORK` は, **<インストールディレクトリ>**`/etc/stagent.env` の中で定義される.
- `$SHARETASK_HOME` は, **<インストールディレクトリ>**`/bin_agent/stagent.sh` の 11 行目付近で定義される (近日, 改善予定).
- #以降はコメントとして無視される.

6.3 エージェント設定ファイルの作成

エージェントコントロール画面で, エージェント設定ファイルを作成するには, `new/edit` ボタンを使います. 設定ファイル欄に作成しようとする設定ファイルの名前をタイプしたのち, `new/edit` ボタンをクリックします. 次に設定ファイル編集ダイアログが表示されますので, そこで設定ファイルの内容を記述してセーブボタンを押します.

もし, 既存の設定ファイルがあり, それをコピーして新たな設定ファイルを作成する場合は, 次のようにします.
画面下段のエージェントの一覧表の設定ファイル欄のファイル名をクリックすると, 画面上段の設定ファイル欄にファイル名がコピーされます. `new/edit` ボタンをクリックすると編集ダイアログが表示されます. このダイアログ上部にあるファイル名欄の値を編集してセーブすれば, そのファイル名の設定ファイルが作成できます.

既存の設定ファイルを呼び出すには, 画面上段の設定ファイル欄にアスタリスク * をタイプするか, ダブルクリックするとファイル名が一覧で表示されますので, そこから選択することでも可能です.

set

設定ファイル欄に表示されている設定ファイルを, フィルター条件に合致するエージェントに反映する.

view

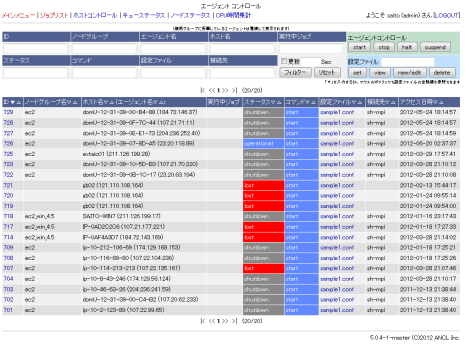

図 6.3.1　エージェントコントロール画面

図 6.3.2　エージェント設定ファイルを編集するボタン群

設定ファイル欄に表示されている設定ファイルの中身をダイアログで表示する.

new/edit

設定ファイル欄に表示されている設定ファイルを作成あるいは編集する. 既存であれば, その内容を編集するダイアログが表示され, 既存でなければ中身が空の編集ダイアログが表示されます.

delete

設定ファイル欄に表示されている設定ファイルを削除します. 削除した場合, 元には戻せませんので注意が必要です.
設定ファイル欄にアスタリスク * をタイプするか, そのテキストボックス上でダブルクリックすると, 既に登録されている設定ファイルの一覧がポップアップします. この一覧から目的のファイルをクリックすれば, 設定ファイル欄の値として反映されます.

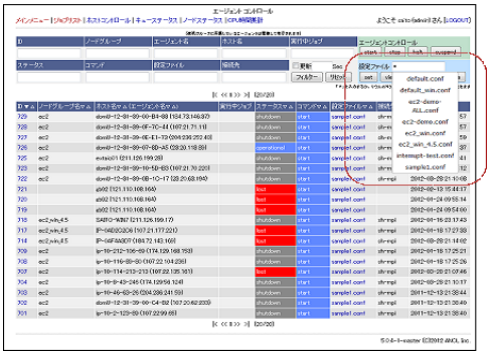

図 6.3.3　エージェントコントロール画面 / 設定ファイル

7　エージェントの起動・停止

計算ノードに常駐する ShareTask エージェントは, OS 起動時に自動的に起動し常駐するように設定されています．その run script は, `/etc/init.d/sharetask-agent` であり, `chkconfig` コマンドで, マルチユーザーレベル init 3 で呼び出されるように設定されています. 確認するには, 以下のコマンドを実行します.

```
chkconfig --list sharetask-agent
```

7.1　エージェントを終了させる

エージェントをバージョンアップする場合には, 常駐しているエージェントのプロセス (Java) を終了させなければなりませんが, それには以下のコマンドを実行します.

```
/etc/init.d/sharetask-agent stop
```

あるいは

```
/sbin/service sharetask-agent stop
```

7.2　エージェントを起動する

エージェントを起動するには, 以下のコマンドを実行します.

```
/etc/init.d/sharetask-agent start
```

あるいは

```
/sbin/service sharetask-agent start
```

エージェントが正常な終了手順を経ずに失われた場合, エージェントの二重起動を抑止するためのロックファイル機構のロックファイルが残存しているため, エージェントの起動に失敗することがあります. その場合は, 以下のコマンドを実行してロックファイルを強制的に削除した上で, エージェントを起動できます.

```
/etc/init.d/sharetask-agent restart
```

あるいは

```
/sbin/service sharetask-agent restart
```

このコマンドは, 現在実行中のエージェントが存在していれば, それを終了させた上で新たなエージェントを起動しますので, エージェントが多重に常駐することはありません.

7.3 エージェントの起動時のパラメーター設定

エージェント起動に関わる環境設定は, 下記ファイルに記述されています.

```
/usr/local/sharetask/etc/stagent.env
```

このファイルでは, 以下の変数が設定されます. 必要に応じて修正してください.

SHARETASK_HOST
標準値: `${SHARETASK_HOST:-$(hostname -s)}`
説明: ホスト名 (サーバーの画面上で表示される名前)

SHARETASK_WORK
標準値: `${SHARETASK_WORK:-/var/spool/sharetask}`
説明: エージェントの稼働ディレクトリのベースパス. これをベースパスとして, その配下に以下のディレクトリを作成する.
auth: エージェント・サーバー間の認証情報
jobs: ジョブ実行時のテンポラリディレクトリのベースディレクトリ
この配下にジョブ ID を名前としてディレクトリを作成する.

SHARETASK_AGENT_OPT
標準値: `"--no-cert-verify -D"`
説明: エージェントの起動時オプション
`--no-cert-verify` は, サーバー証明書の検証を省略する
`-D` は, ログファイルのログレベルをデバッグにする

SHARETASK_JVM
標準値: `${SHARETASK_JVM:-/usr/bin/java}`
説明: エージェントを実行する Java のコマンドパス

SHARETASK_JAVA_OPT
標準値: `""`
説明: エージェントを実行する Java の起動時オプション

8 ジョブを監視する

すべてのジョブ (すべてのユーザーのすべてのジョブ) の一覧を表示するには, トップ画面 (メインメニュー) の左ペインにある **監視・統計** セクション中の**ジョブリスト**をクリックします.

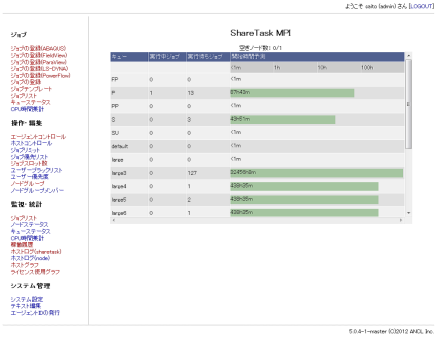

図 8.0.1 トップ画面

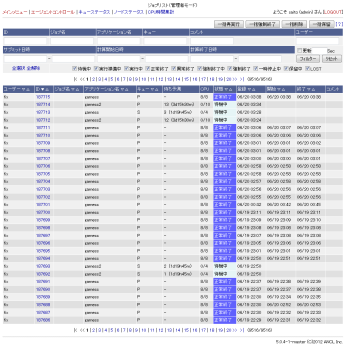

図 8.0.2 管理者専用のジョブリスト画面

ページタイトルが「**ジョブリスト (管理者モード)**」であることにご注意ください. 一般ユーザーのジョブリストでは, 各自のジョブしか閲覧できませんが, 管理者のジョブリストでは,

すべてのユーザーのすべてのジョブが表示されます. ページのデザインが酷似しているため, 勘違いしやすいので注意が必要です. 一般ユーザー用のジョブリスト画面にはない「**ユーザー**」の列があります.

8.1 ジョブの絞り込みフィルター

ジョブを絞り込んで表示するために, 各種フィルターが用意されています.

ID
ジョブ ID で絞り込みます. ハイフンで 2 つのジョブ ID を並べると, 範囲を指定できます.

ジョブ名
ジョブの名前を正規表現で指定できます.

アプリケーション名
アプリケーションの名称 (`progs.conf` で指定された名称) で指定できます. 正規表現が使えます.

キュー
キューの名称 (`progs.conf` で指定された名称) で指定できます. 正規表現が使えます.

コメント
コメント欄に記述された文字列を指定できます. 正規表現が使えます.

ユーザー
ユーザー名で絞り込めます. 正規表現が使えます.

サブミット日時
ジョブを投入した日時範囲で絞り込めます. 日時の入力欄をクリックするとカレンダーがポップアップしますので, 容易に日時を入力できます.

計算開始日時
ジョブが実行開始した日時範囲で絞り込めます. 日時の入力欄をクリックするとカレンダーがポップアップしますので, 容易に日時を入力できます.

計算終了日時
ジョブが実行終了した日時範囲で絞り込めます. 日時の入力欄をクリックするとカレンダーがポップアップしますので, 容易に日時を入力できます.

8.2 ジョブの状態での絞り込みフィルター

ジョブは, 実行待ち・実行中・終了など各種の状態を取りますが, それらの状態をチェックボックスで選択して, 選択された状態のいずれかに合致したジョブだけに絞り込めます. ジョブの状態は表 8.2.1 の通りです.

表 8.2.1　ジョブの状態

状態	**説明**
待機中	キュー (待ち行列) の上で実行待ちしている状態
実行準備中	計算ノードへの割当が開始された状態. まだ実行開始はしていない.
実行中	実行開始された状態
正常終了	実行が正常に終了した状態. ここで正常とは, アプリケーションプロセスの終了ステータス (`exit status`) がゼロであったことを意味する.
異常終了	実行が終了したが異常が検出された状態. ここで異常とは, アプリケーションプロセスの終了ステータス (`exit status`) がゼロではなかったことを意味する.
強制終了中	実行中のジョブが強制終了処理に入ったことを意味する. まだ強制終了が完了していない状態.
強制終了	強制終了が完了し, 当該ジョブに関連するすべてのプロセスが終了した状態.
一時停止中	割り込み実行制御モードにおいて, より高い優先度のジョブがキューに投入されたことにより, 実行をサスペンドして CPU を明け渡した状態.
保留中	キュー (待ち行列) の上で実行待ちしているが, 実行対象からは除外されている状態
LOST	ジョブの状態が不明であることを意味する. 正常系ではあり得ない.

状態遷移図は, 図 8.2.1 の通りです.

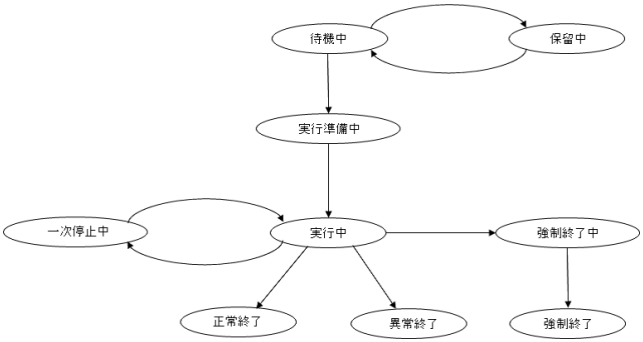

図 8.2.1　状態遷移図

9　ジョブの優先度制御

投入 (登録) されたジョブは, 原則としては投入された順番に実行されますが, 計算資源の平等分配 (フェアシェア) の観点から, 過去一定期間に使用した計算時間 (elapsed time) に基に算出したジョブ優先度によって, ジョブの実行順番を組み替えます.

この優先度を「ユーザー優先度」と呼びます. 過去どこまで遡って各ユーザーの計算時間 (累積) を算出するかは, システム設定画面の次のパラメーターで制御されます.

表 9.0.1　優先度順位パラメーター

パラメーター	**意味**	**内容**
12	優先度計算を使用する	チェックすると有効
13	優先度計算の期間 (単位:日)	30

更新された最新のユーザー優先度は, メインメニューの「**ユーザー優先度**」画面 (図 9.0.1) で確認できます.
優先度は, 0〜100 の整数値をとります.

サーバーバージョン 5.1.1-4 以前では, この優先度の下限値が設定できないために, 優先度が下がりすぎる状況が起こりえます. ユーザー優先度が 20 を下回ると, 当該ユーザーのジョブは極めて実行されにくくなります. 暫定的にこの状況を解決するには, ［**優先度**］に 50〜80

図 9.0.1　ユーザー優先度画面

程度の値を入力し，［**更新ロック**］にチェックを入れ，［**更新**］ボタンを押してください．これによって，設定した優先度に固定され，当該ユーザーのジョブが実行されない状況を回避できます．

10 ログインユーザーの登録

10.1 PAM の場合

ローカルパスワードファイルでユーザー管理をしている場合は, PAM を指定します.

10.2 NIS の場合

ユーザー認証ディレクトリサービスとして NIS を利用している場合, NIS に登録されているユーザーは, そのユーザー名とパスワードで ShareTask サーバーにログインできます.

10.3 LDAP(ActiveDirectory) の場合

ShareTask のログイン認証系を, ActiveDirectory 認証系 (AD) などと LDAP プロトコルで接続している場合, その LDAP 設定は, 下記設定ファイル (表 10.3.1) に記述されるパラメーターで行っています.

```
/home/sharetask/sharetask_server/conf/system.conf
```

表 10.3.1 `system.conf` の LDAP 関連パラメーター

パラメーター	意味
`AUTH=ldap`	LDAP を選択
`LDAP_TYPE=user`	user 方式を選択
`LDAP_HOST=ldap.anclab.com`	LDAP サーバーアドレス
`LDAP_SSL=1`	暗号化通信に SSL を使用
`LDAP_TLS=0`	暗号化通信に TLS は不使用
`LDAP_SASL=0`	暗号化通信に SASL は不使用
`LDAP_ATTR_ID=cn`	ログイン名が格納されている属性名
`LDAP_ATTR_PASSWORD=userPassword`	パスワードが格納されている属性名
`LDAP_BASE_DN=ou=ancluser,dc=anclab,dc=com`	ドメイン名
`LDAP_FILTER=`	フィルターは不使用
`LDAP_ATTR_MAIL_ADDR=mail`	メールアドレスが格納されている属性名

10.4 ホワイトリストへの登録方法

ShareTask のログイン認証系を NIS・LDAP に接続した場合, それらディレクトリサービスに登録されているすべてのユーザーが ShareTask にログインできてしまいます. そこで, ShareTask にログインできるユーザーを限定するフィルターのしくみが必要になります. このしくみをホワイトリストと呼びます. ホワイトリストのユーザーを記述するファイルは, `login.acl` です.

```
/home/sharetask/sharetask_server/conf/acl/login.acl
```

このファイルには, ユーザー名について正規表現を使用してログインの可否を記述します.

```
[ +|-]:user_name_regexp
```

例 :

`+:^u0001$`	u0001 というユーザーはログイン可 (+)
`+:^u0002$`	u0002 というユーザーはログイン可 (+)
`-:.*`	以上以外の全てのユーザーはログイン不可 (-)

10.5 ユーザー ID の確認方法

管理者メニューのシステム設定画面において, 管理者ユーザーを登録するにはユーザー ID が必要です. このユーザー ID を知るには, ブラウザで下記 URL に直接アクセスしてください.

`http://`**サーバー**`/admin/usersdb.cgi`

11 ライセンス使用履歴のグラフ化

ライセンス使用履歴を取得するためのスクリプト群は,

```
$SERVER_HOME/cgi-common/bin
```

にあります.

get_license.sh

取得したいすべてのフィーチャーについて記述します.
フィーチャー名とグラフ表示上の名称との変換もこの中で行います.
グラフ化したいフィーチャーを追加する場合は, このスクリプトを編集します.

get_license.pl

上記 `get_license.sh` を呼び出し, その出力を解析することにより, 各フィーチャーについてライセンス数を取得します. 通常は, このスクリプトを編集する必要はありません. この perl スクリプトは, サーバー内部から呼び出されます.

12 バックアップの取り方

ShareTask サーバーマシンには, ユーザー, ジョブ, システムのさまざまな情報がデータベースとファイルシステムに分かれて保管されています. したがって, バックアップ手順は, データベースのバックアップと, ファイルシステムのディレクトリ階層のバックアップの 2 つに分かれます.

12.1 データベースのバックアップ

ShareTask サーバーには, ユーザー, ジョブ, システムのさまざまな情報がデータベース (PostgreSQL) で保管されています. 定期的にバックアップすると安心です. バックアップには, Linux で以下の操作を行います (いずれも root 権限が必要です).

```
# pg_dumpall -U postgres | gzip > backup_file.gz
# tar cvzf backup_file.tar.gz /var/www/html/stask/files
```

また, バックアップタイミングで, 以下の操作も行ってください.

```
# vacuumdb -az -U postgres
# reindexdb -a -U postgres
```

12.2 ディレクトリのバックアップ

バックアップするべきディレクトリは, 次の表 12.2.1 の通りです.

表 12.2.1 バックアップするディレクトリ

ディレクトリ	説明
サーバーのインストールディレクトリ	/home/sharetask/sharetask_server/配下を tar でとります.
エージェントのインストールディレクトリ	/app/sharetask/配下を tar でとります.
データベース	pg_dumpall コマンドを用いてダンプ形式でバックアップをとります.

一連のバックアップ手順は, /home/sharetask/Backup.sh というスクリプトにまとめられています.

12.3 cron によるバックアップの自動化

このスクリプトは, crontab に登録されています.

```
01 22 * * 7 /home/sharetask/Backup.sh >>
                                /data/sharetask/backup/Backup.log  2>&1
```

crontab の設定を確認するには, root で以下のコマンドを実行します.

```
# crontab --l
```

crontab の設定を修正するには, root で以下のコマンドを実行します.

```
# crontab -e
```

13 ShareTask のバージョンアップ手順

13.1 サーバーのバージョンアップ

ShareTask サーバーの新たなバージョンを導入する手順は, 以下のとおりです.

1. Apache を停止します.
   ```
   # /sbin/service httpd stop
   ```
2. sharetaskd を停止します.
   ```
   # /sbin/service sharetaskd stop
   ```
3. パッケージを展開します.
   ```
   # #cd /home/sharetask
   # tar xzf - < sharetask_server-{NEWVER}
   ```
4. 現行のカスタマイズ部分を反映します.
   ```
   # mv sharetask_server-{NEWVER}/conf
     # sharetask_server-{NEWVER}/conf.ORG
   # mv sharetask_server-{NEWVER}/cgi-common/bin
     # sharetask_server-{NEWVER}/cgi-common/bin.ORG
   # cp a sharetask_server-{OLDVER}/conf
     # sharetask_server-{NEWVER}
   # cp a sharetask_server-{OLDVER}/cgi-common/bin
     # sharetask_server-{NEWVER}/cgi-common
   ```
5. シンボリックリンクをはりかえます.
   ```
   # rm sharetask_server
   # ln s sharetask_server-{NEWVER} sharetask_server
   ```
6. Apache を起動します.
   ```
   # /sbin/service httpd start
   ```
7. sharetaskd を起動します.
   ```
   # /sbin/service sharetaskd start
   ```

以上でサーバーのバージョンアップは完了です.

13.2 エージェントのバージョンアップ

エージェントの新たなバージョンを導入する場合は, 以下の手順で行います.

1. エージェントを停止します.

```
# /sbin/service sharetask-agent stop
```

2. パッケージを展開します.

```
# cd /usr/local
# unzip q sharetask-client-{NEWVER}
```

3. 現行設定をコピーします.

```
# mv sharetask-client-{NEWVER}/conf
  sharetask-client-{NEWVER}/conf.ORG
# mv sharetask-client-{NEWVER}/etc
  sharetask-client-{NEWVER}/etc.ORG
# cp a sharetask-client-{OLDVER}/conf
  sharetask-client-{NEWVER}
# cp a sharetask-client-{OLDVER}/etc
  sharetask-client-{NEWVER}
```

4. シンボリックリンクをはりかえます.

```
# rm sharetask
# ln s sharetask-client-{NEWVER} sharetask
```

5. 初期設定を行います.

```
# cd sharetask
# make install_agent install_command
```

6. エージェントを起動します.

```
# /sbin/service sharetask-agent start
```

以上でエージェントのバージョンアップは完了です.

14 アプリケーションの追加手順

14.1 概要

新しいアプリケーションを ShareTask に登録するには, 3 つの部分の設定ファイルの編集が必要です.

1. ラッパースクリプト (`$AGENT_HOME/appl`)
2. エージェントの設定ファイル (エージェントコントロール画面)
3. ジョブ登録画面のメニュー設定ファイル (`$SERVER_HOME/conf/progs.conf`)

`$AGENT_HOME, $SERVER_HOME` は, それぞれエージェントとサーバーのインストールディレクトリです. 標準では, 次のとおりです.

```
$AGENT_HOME=/usr/local/sharetask
$SERVER_HOME=/home/sharetask/sharetask_server
```

14.2 プログラム名とプログラム実体の結びつけ

以上 3 つの編集箇所は, プログラムの名前 (ユーザーが命名する任意の名前. 以下プログラム名と呼ぶ) と, それに対応するプログラム実体 (コマンド) が以下の流れで結びつけられているためです.

プログラム名からラッパースクリプトまでの結びつきは, 次のとおりです.

1. ジョブのアプリ名属性が, ジョブ登録画面のプルダウンメニュー (プログラム名) で与えられる. プルダウンメニューの選択肢は, `$SERVER_HOME/conf/progs.conf` に記述される.
2. エージェントは, プログラム名をキーにしてサーバーにジョブを問い合わせる. プログラム名は, エージェント設定ファイルの `define_program{name}`の値として記述される.
3. エージェントは, サーバーから払い出されたジョブを設定ファイルに記述されたラッパースクリプトを呼び出すことにより, 該当するプログラムで実行する. ラッパースクリプトのファイルパスは, `define_program{wrapper, executable}` で記述される.

14.3 ラッパースクリプトの構成

ラッパースクリプトは 2 段階, 2 つのスクリプトから構成されています.

第 1 段階　define_program{wrapper}で指定されたラッパースクリプト
第 2 段階　define_program{executable}で指定されたラッパースクリプト

第 1 段階では, 個々のプログラムに依存しない共通の前処理と後処理を行います. 第 2 段階では, プログラム固有の前処理ののち, プログラム実体 (コマンド) を呼び出し, その実行終了ののち後処理を行います. したがって, 一般的な構成方法としては, 以下のような関係になります.

エージェント設定ファイル

```
define_program{
   name = myprog1
   wrapper = /usr/local/sharetask/appl/jobwrapper.sh
   executable = /usr/local/sharetask/appl/myprog1_execute.sh
}

define_program{
   name = myprog2
   wrapper = /usr/local/sharetask/appl/jobwrapper.sh
   executable = /usr/local/sharetask/appl/myprog2_execute.sh
}
```

呼び出しフロー

AGENT → jobwrapper.sh → myprog1_execute.sh → myprog1
または
AGENT → jobwrapper.sh → myprog2_execute.sh → myprog2

14.4 バージョンの異なるプログラムを追加する場合

バージョンは異なるが, すでに同じプログラムについて設定が行われている場合は, ラッパースクリプトファイルを新たに作成する必要はありません. progs.conf を編集して, 新たなバージョンについてプログラム名を定義します.

progs.conf **の例**

abaqus_6.10-2 ⇒ Q[1,2,4]
abaqus_6.11-1 ⇒ Q[1,2,4]

これに対応するように, エージェント設定ファイルでも, その新しいプログラム名をジョブポーリングのキーとなるように記述します.

エージェント設定ファイルの例

```
define_program {
   name = abaqus_6.10-2
   name = abaqus_6.11-1
   version = Q
   workdir_parent = /var/spool/sharetask/jobs
   executable = /usr/local/sharetask/appl/abaqus_execute.sh
   wrapper = /usr/local/sharetask/appl/jobwrapper_abaqus.sh
}
```

ひとつの define_program ブロックの中では, 複数の name を定義できます. この場合, abaqus_execute.sh には, SHARETASK_PROGRAM_NAME という環境変数としてプログラム名が継承されます. この環境変数を参照することで, 実行すべきプログラム実体 (コマンドパス) を決定できます.

ラッパースクリプトの例

```
case $SHARETASK_PROGRAM_NAME in
abaqus_6.10-2)
   /opt/abaqus/Command/abq6102 …
   ;;
abaqus_6.11-1)
   /opt/abaqus/Command/abq6111 …
   ;;
esac
```

14.5 ラッパースクリプトに継承される環境変数

このように, 第 2 段階のラッパースクリプトには, 環境変数によってさまざまなジョブの属性値が与えられています (表 14.5.1). これら環境変数を定義するのは, 第 1 段階のラッパースクリプトの役目です. これら環境変数を定義した上で, 第 2 段階のラッパースクリプトを呼び出します. 第 2 段階のラッパースクリプトは, これらを参照することによって実行すべきプログラム本体のパスを決定することをはじめ, さまざまな処理を施せます.

表 14.5.1 環境変数とジョブの属性値

環境変数	属性値
SHARETASK_JOB_ID	ジョブ ID
SHARETASK_NP	並列度 (CPU 数)
SHARETASK_PWD	実行ディレクトリパス
SHARETASK_OUTDIR	制御情報ファイルの生成ディレクトリパス
SHARETASK_SYS_WORK_DIR	ジョブ制御用テンポラリディレクトリパス
SHARETASK_USER	ジョブのユーザー名
SHARETASK_USER_HOME	ジョブのユーザーのホームディレクトリ
SHARETASK_TIME_LIMIT	ジョブの強制打ち切り時間
SHARETASK_STDIN	標準入力につなぐファイルパス
SHARETASK_STDOUT	標準出力につなぐファイルパス
SHARETASK_STDERR	標準エラー出力につなぐファイルパス
SHARETASK_COMMAND	第 2 段階のラッパースクリプトを呼び出すコマンドライン
SHARETASK_HOST_LIST	ホストと CPU 数のリスト (SHARETASK_HOSTS が指すマシンファイルと形式は異なるが同一内容)
SHARETASK_HOSTS	MPI 用マシンファイルのパス
SHARETASK_PROGRAM_NAME	プログラム名
SHARETASK_QUEUE_NAME	キュー名

15 アプリケーションライセンス検査

15.1 はじめに

アプリケーションベンダーから提供されるライセンスで, 同時実行数が制御されているタイプのアプリケーション (たとえば ABAQUS など) を利用する場合は, 待ち行列から実行するジョブを選択する際に, 当該ジョブのアプリケーションライセンスが必要数取得できるか (空いているか) を考慮しなければなりません. アプリケーションライセンスが不足している状況下でジョブを実行しようとすると, ジョブが異常終了したり, ライセンスが取得できるまで計算ノードを占有したままとなり, 計算資源の無駄遣いが発生します.

15.2 UserDefine.filter

ひとつのジョブが必要とするアプリケーションライセンス数の決定は, ジョブの並列度の他, アプリケーションベンダーのライセンス方式に依存して, さまざまな計算式があり多様性があります. ShareTask では, この多様性に対応するために, ライセンス数が取得可能かを判定するロジックをスクリプトでプログラミングできるようになっています. この仕組みは, `UserDefine.filter` と呼ばれる

```
/home/sharetask/sharetask_server/plugin
```

に配置された一式の Perl スクリプト群です. これらのスクリプトは, アプリケーションの種類, ユーザー環境, 運用ポリシーに適応できるようにユーザーがカスタマイズすることを想定したつくりになっています.

この `UserDefine.filter` に記述された判定ロジックは, ShareTask 内部で待ち行列から実行対象のジョブを拾い出すたびに呼び出されますので, その時に `UserDefine.filter` がジョブの並列度などの属性値から必要ライセンス数を計算し, ライセンスマネージャーに問い合わせて得たライセンス残数とから, 真偽値 (アプリケーションライセンスが満たされるか否か) を適切に返すことによって, ShareTask 内部の実行対象選択を制御できます.

`UserDefine.filter` を構成するファイルを表 15.2.1 にまとめます. また, ファイル相互の関係を図 15.2.1 に示します. ShareTask 内部から呼び出されるのは, `UserDefine.filter` というファイルに記述された Perl スクリプトです. これを起点に, アプリケーション個別のライセンス判定が行われます.

表 15.2.1　UserDefine.filter を構成するファイル

ファイル名	説明
UserDefine.filter	ShareTask 内部 (コア) から呼び出される Perl スクリプトであり, 以下に説明するスクリプトの呼び出し連鎖の最上位に位置する. / ライセンスチェック対象のアプリケーションを追加するにはこれを編集して, 新規作成した Check{APPL}Liscense オブジェクトを CheckSystem オブジェクトに登録する.
CheckSystem.pm	ライセンス検査を統括するクラスであり, UserDefine.filter から呼び出されて, 登録された Check{APPL}License オブジェクトのライセンス検査関数を呼び出す.
check.conf	基本的な動作パラメーターを記述したファイル
except_lic.conf	ライセンス残数がゼロになるまで使うのではなく, 残数に余裕をもたせる設定を記述するファイル
Conf.pm	以上の設定ファイルを読み込むためのクラス
CheckDB.pm	ShareTask 内部のデータベースから情報を抽出するための関数をまとめたクラス
Job.pm	ジョブの属性値 (ジョブ ID, 状態, プログラム名, キュー名など) をまとめたデータオブジェクト
Check{APPL}License.pm	アプリケーション特定のライセンスを検査するクラス
GetLicenseFor{APPL}.pl	ライセンスマネージャーへの問い合わせ処理を記述するライセンスマネージャーのホスト名, ポート番号, アプリケーションのフィーチャー名を環境にあわせて編集する.

15.3　アプリケーションの追加

新たなアプリケーションについてライセンス検査を追加する手順は, 次のとおりです.

1. Check{APPL}License.pm ならびに GetLicenseFor{APPL}.pl のペアを, 既存の ABAQUS, あるいは LS-DYNA 用のファイルをひな形として作成し, ライセンスマネージャーの IP アドレス, ポート番号, アプリケーションの feature の名称を設定します.
2. UserDefine.filter を修正して, Check{APPL}License オブジェクトが登録される

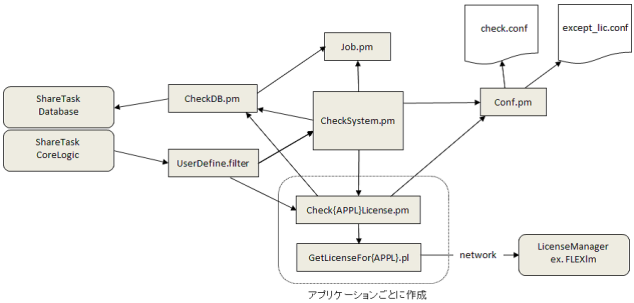

図 15.2.1　UserDefine.filter のファイル相互の関係図

ようにします. 具体的には, 以下の下線部分を追加します.

```
use lib '/home/sharetask/sharetask_server/conf/plugin';
use CheckDB;
use CheckSystem;
use CheckAbaqusLicense;
use CheckLsdynaLicense;
use CheckNewLicense;

my $cdb=CheckDB->new();
my $cs=CheckSystem->new($job_id,$agent_id,$cdb);
$cs->addCheck(CheckAbaqusLicense->new($cdb));
$cs->addCheck(CheckLsdynaLicense->new($cdb));
$cs->addCheck(CheckNewLicense->new($cdb));
```

15.4　ABAQUS トークンの算出

既存の CheckAbaqusLicense.pm の中では, 以下のようにトークン数を計算しています. 必要に応じて修正してください. また, 対象となる ABAQUS ジョブは, プログラム名が abaqus で始まるジョブで認識しています.

CheckAbaqusLicense.pm の部分

```
# GPU を使用しないジョブ
($ncpu == 2 ) ? 6 :
($ncpu == 4 ) ? 8 :
($ncpu == 8 ) ? 12 :
($ncpu == 16 ) ? 16 :
# GPU を使用するジョブ
($ncpu == 2 ) ? 7 :
($ncpu == 4 ) ? 9 :
($ncpu == 8 ) ? 12 :
($ncpu == 16 ) ? 16 :
```

15.5 同時ジョブ実行数によるチェック

Dytran, Stream9, ADVC については, 同時ジョブ実行数制限という形でライセンス検査を行っています. 制限値は, prog.conf ファイル内でプログラム名, 制限値の組 (空白文字区切り) で定義され, 現設定は以下の通りです.

= prog.conf =

```
Dytran      1
Stream9     2
ADVC        2
```

15.6 ユーザーあたりの同時実行ジョブ数

全てのキューについて, 1 ユーザーが同時に実行可能なジョブ数を 1 とする制限を設けています. 本制限は, 1 ユーザーが同時実行可能なジョブ数の上限をキューに施すことで実現されており, user.conf ファイルで制限値を定義 (キュー名, 制限値の組) しています.

= user.conf =

```
para02      1
para04      1
para08      1
para16      1
para02g     1
para04g     1
para08g     1
para16g     1
```

15.7 ログメッセージ

UserDefine.filter のログメッセージ (デバッグ情報) は, /var/log/httpd/error_log に出力されます. そのログレベル (メッセージの詳細度) は, check.conf ファイルで

DEBUG=**レベル値**

の形で定義されます. レベル値は 0~5 の整数値で, 0 では全くログが出力されず, 高い値になるにつれて詳細なメッセージが出力されます. 高い値では膨大なメッセージが出力され, ログファイルが肥大化するだけでなく, システムのパフォーマンス劣化を引き起こしますので注意してください. 通常の運用では 1 を推奨します.

以上