コードレビュー Vol.5

CODE Review

USP 研究所技術研究員
written by 大内智明

今回は6、7月号と続けて取り上げたテーマ「集中して発生するトランザクションの処理方法」に関する記事の最終回となります。ユニケージ開発手法における配信用LV3データを他サーバーへ配信する処理についてご説明します。

ポータルサーバーへ配信

まずは、本ページの図 1 をご覧ください。本部や店舗で発生したトランザクションを集積して、LV2 を作成する処理を前回説明しました。今回は、加工された配信用データ（→補足 1）を各店舗や本部の担当者が閲覧できるようにポータルサーバーに配信する処理部分について説明します。

配信部分の特徴は、各業務サーバーで発生したトランザクションを集積（定期的にファイルにまとめる）、加工（配信用ファイルとしてレイアウト）した後に、加工したファイルを処理結果を表示するポータルサーバーに送信する点にあります。

【補足 1】トランザクション（LV1）は各業務サーバーで発生した生データです。そのままの状態では配信元の各業務サーバーにおいて使用しやすいレイアウトにとどまります。そこで、加工処理を行って、配信先で使用するのに適したレイアウトすなわち配信用データ（LV3）に変換しています。

技術的な概要

配信の方法はディレクトリ単位ではなく、ファイル単位で配信先サーバーを決めて行われます（→補足 2）。ただし、配信をファイル単位で行うと、ファイルと配信先サーバーの組み合わせパターンが大きく増えることがあります。そこでユニケージ開発手法では、配信する件数が多い場合に、処理を「並列＆バックグラウンド」で実行させることで解決しています（図 2）（→補足 3）。

図 1　常駐型シェルスクリプトの処理フロー

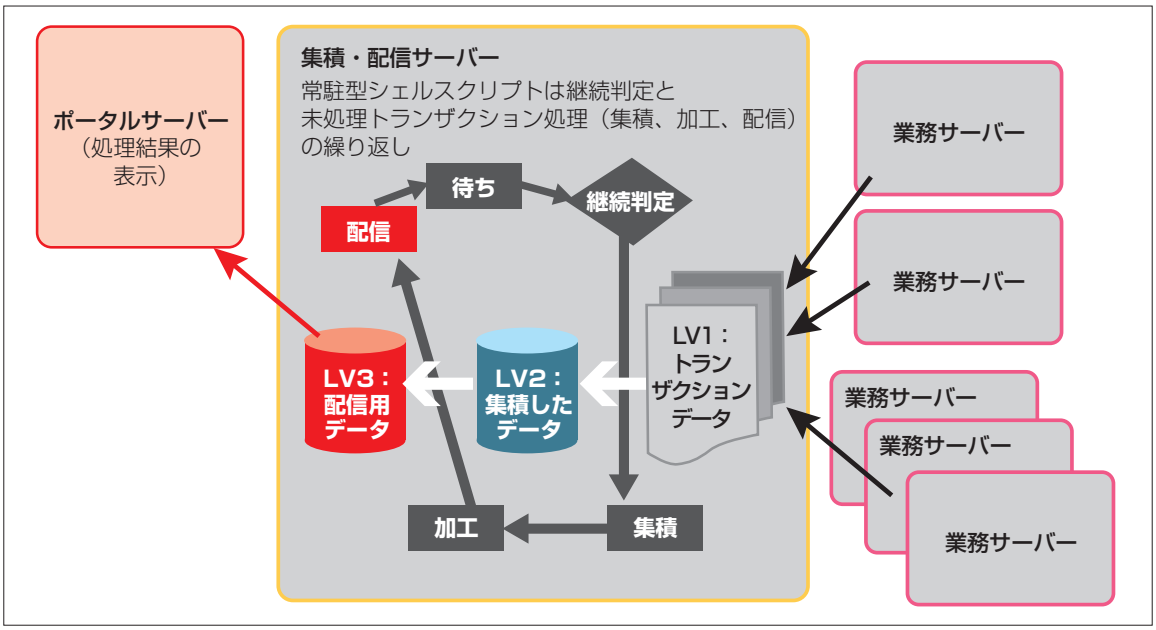
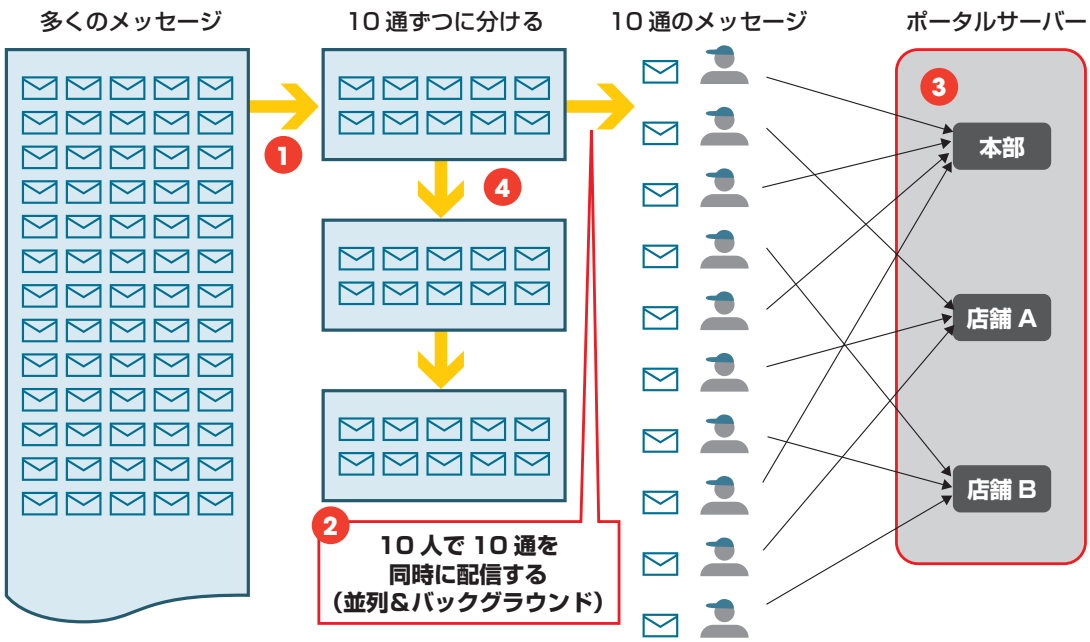


図2　並列&バックグラウンド処理のイメージ



❶多くのデータ（メッセージ）を10通ずつに分ける
❷それぞれのメッセージを10名の配達員が同時に宛先に届ける。
❸❷の配達処理がすべて終了するまで待つ（waitコマンド）
❹次の10通のメッセージを配達する（❷、❸を繰り返す）

並列&バックグランドで処理する効果は、トータルの処理時間を短縮できることです。1行ずつ処理せず、複数の処理を並列で行います。並列処理を適用できる条件は、「同じ処理の繰り返し」であることです。今回は、ファイル送信の繰り返しなので適用することが可能です。なお、並列処理では、順番通りに処理が終了しない点に注意します。決められた順番で処理を終了させる必要がある場合には、並列処理は向きません。また、処理を複数実行するため、バックグラウンド処理を行うことが必須となります。

【補足2】ディレクトリ単位でまとめて配信する場合には、ディレクトリ内に配信先で使用しないファイルが含まれるケースが出てきます。ファイル単位で送信すれば、不要なファイルを配信しなくてよくなり、ネットワークにも不要な負荷をかけなくて済みます。

【補足3】「並列&バックグラウンド」とは、繰り返し実行する処理をバックグラウンドで複数実行するという意味です。

コードの見どころ

今回のソースコードは、配信処理について記載しています。このコードは、ユニケージ開発手法の特徴の一つである「上から下に順番に読めるコード」になっているので、説明も上から順番にしていきます。

［1］配信元データと配信先サーバーの一覧から送信リストを作成する（79～90行目）（画面1）。

ヒアドキュメントで配信先グループ（業務ごとに分けられたサーバー群）と配信するファイルの種類（→補足4）の一覧を定義します。次に配信先グループをmsctrlコマンドで展開して、配信先サーバー／配信するファイルの種類／配信するファイルの日付の送信リストを作成します。

［2］作成した送信リストは、並列&バックグラウンド処理に適した形にするため、10行ずつのリストを保持するファイルに分割します（96、97行目）。

［3］分割したリストは、並列&バックグラウンドで処理します（100～122行目）（→図2）。

【補足4】「HONBU（本部）」は、ファイルの種類を表します。例えば本部向けと店舗向けの発注データが存在する場合に、それが本部向けのデータであることを示します。

リスト１　集積処理を行うプログラム

```
#!/bin/ush -xve
# システム名　　：ＵＳＰシステム
# サブシステム名：連絡アラートの配信(日次)
# 業務名　　　　：連絡アラート
# プログラム名　：連絡アラートＬＶ３配信(日次)
# 備考(Usage)　：DISTRI_LV3TBL.TIMES.ALERT_TSUCHI [YYYYMMDD]
# シェル名　　　：DISTRI_LV3TBL.TIMES.ALERT_TSUCHI
# 作成日　　　　：2013/08/02
# 会社名　　　　：USP
# 作成者　　　　：A.Asaba

#//////////////////////////////////////////////////////////////////////////
# 初期設定
#//////////////////////////////////////////////////////////////////////////

# 走行ログの記録
logd="${HOME}/LOG"                                                  # ログディレクトリ
logf="${logd}/LOG.$(basename $0).$(date +%Y%m%d)_$(date +%H%M%S)_$$"  # ログファイル名
echo  "${logf}" > /dev/null 2>&1
log 2> ${logf}

# パスの定義
PATH=/home/UTL:/home/TOOL:/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin:${PATH}
LANG=ja_JP.UTF-8

#--------------------------------------------------------------------------
# 変数の定義
#--------------------------------------------------------------------------
tmp="/tmp/$$-$(basename $0)_$(date +%Y%m%d%H%M%S)"          # 一時ファイル
hostname="$(HOSTNAME)"                                      # サーバー名
semd="${HOME}/SEMAPHORE"                                    # セマフォディレクトリ
rsemd=/home/RCV/SEMAPHORE                                   # 受信セマフォディレクトリ
sday="$(date +%Y%m%d)"                                      # 本日日付セット
myrole="COM"


subd="ALERT "
lv3d="/home/DATA/LV3"
rirekid="${lv3d}/ RENRAKU/${subd}/RIREKI"

# エラー時の終了処理定義 (ushエラーハンドラ)
err ERROR_EXIT(){
  touch ${semd}/$(basename $0).${hostname}.ERROR.${sday}
  echo "${hostname} $(basename $0)_${sday} ERROR $(date +%Y%m%d%H%M%S) ${logf}" >> ${logd}/UPCNT
  exit 1
}

sday=$(date +%Y%m%d)
# 引き数の確認
[ $# -eq 1 ] && sday=$1

# 簡易YYYYMMDD日付チェック
if ! isdate ${sday} ; then
  echo "Parameter DATE error:[${sday}]"
  ERROR_EXIT
fi

# 前回セマフォの消去
rm -f ${semd}/$(basename $0).${hostname}.*.${sday}  >/dev/null 2>&1
```

```
60
61 # 起動時刻の記録
62 echo "${hostname} $(basename $0)_${sday} START $(date +%Y%m%d%H%M%S)" >> ${logd}/UPCNT
63 touch ${semd}/$(basename $0).${hostname}.START.${sday}
64
65 #////////////////////////////////////////////////////////////////////////
66 # 処理部
67 #////////////////////////////////////////////////////////////////////////
68
69 # 正マシンのみ処理
70 if [ "$(msctrl -host ${hostname} -job ${myrole} -print msflg)" == "M" ] ; then
71
72   #////////////////////////////////////////////////////////////////////////
73   # 配信
74   #////////////////////////////////////////////////////////////////////////
75
76   # 配信先ROLEグループの取得
77
78 # 1:roleグループ 2:ファイル種別
79 cat << ETX | self 1/NF > ${tmp}-rolelist
80 MENU HONBU
81 MENU STORE
82 LOGIN HONBU
83 LOGIN STORE
84 ETX
85 cat ${tmp}-rolelist |
86 while read role target; do
87   msctrl -job ${role} -print host |
88   gawk '{print $1,"'${target}'","'${sday}'"}'
89 done |
90 LANG=C sort -u > ${tmp}-sendfilelist
91
92   # 対象がなければ終了
93   if [ -s ${tmp}-sendfilelist ]; then
94
95     # 配信先サーバーの展開リスト作成
96     cat ${tmp}-sendfilelist       |
97     gyocut -10 $tmp-dstlist.host_uniq.%02d -
98
99     # 配信処理をバラで起動(最大10バラ)
100    echo $tmp-dstlist.host_uniq.?? |
101    tarr                          |
102    ugrep -v '¥?'                 |
103    xargs -J % find % -type f     |
104    while read fname; do
105      cat ${fname} |
106      while read sndserver shubetsu targetday; do
107        # ファイル名
108
109        target_file_path=${rirekid}/${shubetsu}.${targetday}.gz
110        (
111        # ファイルが存在すれば処理
112        [ -s ${target_file_path} ] || continue;
113        cp -p ${target_file_path} ${target_file_path}.${$}
114        ssh -n ${sndserver} 'mkdir -p '"${rirekid}"'/'
115        scp -p ${target_file_path}.${$} ${sndserver}:${target_file_path}.${$}
116        ssh -n ${sndserver} 'mv '"${target_file_path}.${$}"' '"${target_file_path}"''
117        rm -f ${target_file_path}.${$}
118        ) < /dev/null &
```

サーバー管理ファイルと msctrl コマンドについては画面 1 を参照

[1] 送信リストの作成 (79 ～ 90 行目)

[2]10 件ずつ分割する処理 (96 ～ 97 行目)

gyocut については画面 2 を参照

[3] バックグラウンドで送信処理 (100 ～ 122 行目)

最大 10 件の対象ファイルをバックグラウンドで送信処理する (110 ～ 118 行目)。

```
119        done
120        wait
121        sleep 1
122      done
123    fi
124
125 fi
126
127 #////////////////////////////////////////////////////////////////////////
128 # 終了処理
129 #////////////////////////////////////////////////////////////////////////
130 # 終了時刻の記録
131 echo "${hostname} $(basename $0)_${sday} END $(date +%Y%m%d%H%M%S)" >> ${logd}/UPCNT
132 touch ${semd}/$(basename $0).${hostname}.END.${sday}
133
134 # 終了
135 rm -f ${tmp}-*
136 echo "$(basename $0) exit 0"
137 exit 0
```

wait コマンドでバックグラウンド起動したプロセスが終了するまで待機させる。

画面 1 サーバー管理ファイルと msctrl コマンド

```
$ cat MSCTRLTBL ↵
host role msflg ctrl
menu1 MENU M C
menu2 MENU M C
····
menu30 MENU M C
$ msctrl -job MENU -print host ↵
menu1
menu2
·····
menu30
```

サーバー管理ファイル

この例では、msctrlコマンドを用いて、同じ役割(ここでは、MENU)に属するホスト名をすべて抽出している。

各サーバーの管理は、役割、正副、管理可否などの情報を盛り込んだMSCTRLTBLファイルを用いて行う。MSCTRLTBLは４列からなるデータ。１列目の host とはサーバーのホスト名、２列目の role は業務種類別にグループ化されたサーバーの役割を示す。３列目の msflg は、M:正（稼働）、S：副（待機）、B：負荷分散のいずれの状態にあるかを表している。４列目の ctrl は、C：管理（動作中）、-：管理外（動作していない）の状態を示している。

画面 2 gyocut コマンドによって指定行数でファイルを分割する

```
$ gyo data ↵
4500
$ gyocut -1000 file.%02d data ↵
file.01
file.02
file.03
file.04
file.05
```

4500行あるdataファイルを1000行ごとに分割する。
file.01～file.04には1000行ずつ出力され、
最後のファイルfile.05には、余りの500行が出力される。

まとめ

３カ月にわたって見てきた常駐型シェルスクリプトの解説も今回の配信処理でひとまず終了です。常駐型シェルスクリプトの書き方、集積方法、並列＆バックグラウンド処理などをご紹介しましたが、いかがだったでしょうか。開発時の参考になれば幸いです。

次回からは新しいテーマを取り上げる予定です。