国立研究開発法人　理化学研究所　御中

2020年度InSitu処理向け三次元可視化  
フレームワークのプロトタイプ整備

作業報告書

第1.0版

富士通株式会社

2021年1月

改版履歴

|  |  |  |
| --- | --- | --- |
| リリース | 版数 | 備考 |
| 2021/01/27 | 1.0 | 初版 |

# 目次

[目次 3](#_Toc62040332)

[1. はじめに 4](#_Toc62040333)

[2. 実装報告 5](#_Toc62040334)

[2.1. システム設計 5](#_Toc62040335)

[2.1.1. システムの構成 5](#_Toc62040336)

[2.1.2. プログラム間の通信データ 5](#_Toc62040337)

[2.2. モジュール設計 7](#_Toc62040338)

[2.2.1. Temporal Buffer 7](#_Toc62040339)

[2.2.2. TB2C server 10](#_Toc62040340)

[2.2.3. TB2C client 14](#_Toc62040341)

[3. 動作検証報告 21](#_Toc62040342)

[3.1. 概要 21](#_Toc62040343)

[3.2. ITOフロントエンドでの環境構築 21](#_Toc62040344)

[3.2.1. ノード構成 21](#_Toc62040345)

[3.2.2. Spackのインストールおよび設定 21](#_Toc62040346)

[3.2.3. Node.jsおよびnpmのインストール 22](#_Toc62040347)

[3.2.4. Python3およびPythonモジュール群のインストール 22](#_Toc62040348)

[3.2.5. Spack環境のbash設定 22](#_Toc62040349)

[3.2.6. ChOWDERのインストール 22](#_Toc62040350)

[3.2.7. TB2Cのインストール 23](#_Toc62040351)

[3.3. 実施方法 23](#_Toc62040352)

[3.3.2. ITOログインノードでのポートフォワーディング設定 24](#_Toc62040353)

[3.3.3. ローカルPCでのポートフォワーディング設定 24](#_Toc62040354)

[3.3.4. ChOWDERコントローラーの実行 24](#_Toc62040355)

[3.3.5. TB2C clientの実行 24](#_Toc62040356)

[3.3.6. ChOWDERディスプレイのレンダリング時間測定 25](#_Toc62040357)

[3.4. 実施結果 25](#_Toc62040358)

# はじめに

本書は、国立研究開発法人理化学研究所向け「2020年度InSitu処理向け三次元可視化フレームワークのプロトタイプ整備」の作業報告書です。

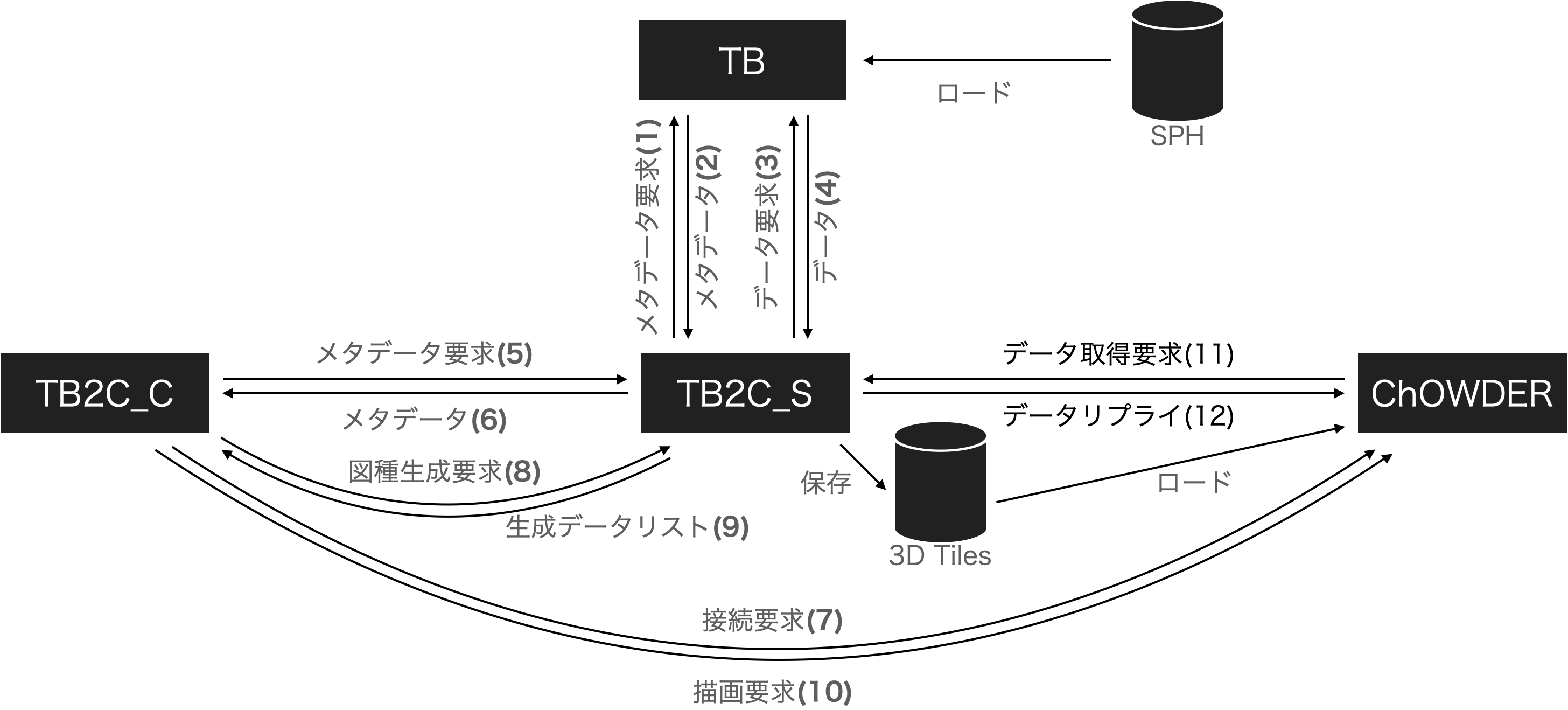
本作業で実装したTB2Cの実装報告および動作検証報告を記述しています。

# 実装報告

# システム設計

# システムの構成

TB2Cは、Temporal Buffer (TB)とTB2C\_server (TB2C\_S)およびTB2C client (TB2C\_C)から構成されるシステムであり、ChOWDERに接続して動作します(下図参照)。



Temporal Buffer(TB)は、SPHフォーマットで用意された時系列の数値シミュレーション結果データを読み込んでバッファリングし、外部からの要求に応じて必要な時刻スライス・物理量のデータを提供します。

TB2C client(TB2C\_C)はユーザーが直接操作するGUIプログラムであり、可視化パラメータの設定や時刻スライスの指定、視点の変更等の操作を行い、ChOWDERに対する表示更新要求を行います。

TB2C server(TB2C\_S)は、TB2C\_Cからの要求に応じて、TBが保持するデータを取得して可視化図種の生成を行い、ChOWDER表示用に3D-Tiles形式のファイルに出力します。

TBおよびTB2C serverは、ChOWDERと同一のマシン上で動作することを前提としています。TB2C clientは別のマシン上で動作し、ChOWDERおよびTB2C serverと通信を行います。

# プログラム間の通信データ

以下に、各プログラム間で通信されるデータを示します。

(1) TB2C\_serverからTBへのメタデータ要求

 TBのポートへのHTTP GET

path="/",

param:なし

(2) TBからTB2C\_serverへ返されるメタデータ

 JSON[{"id":データID,

　　　"uri":先頭SPHファイルのURI,

　　　"type":"SPH",

　　　"dims":[I,J,K],

　　　"datalen":ベクトル長,

　　　"bbox":[[x0,y0,z0],[x1,y1,z1]],

 　　　"steps":ステップ数,

　　　"timerange":[開始時刻,終了時刻],

　　　"vrange":[最小値,最大値] }, …]

TBが保持するデータがリスト化されたJSONが返される(今回の実装では１個のみ)

(3) TB2C\_serverからTBへのデータ要求(指定したタイムステップのデータを取得する)

 TBのポートへのHTTP GET

path="/data",

param:id=データID,

step=取得するステップ番号

(4) TBからTB2C\_serverへ返されるデータ

 JSON{"type":"SPH",

"step":ステップ番号,

"data":{データ本体} }

{データ本体}は、SPHクラスを base64でエンコードしてJSON化したもの

(5) TB2C\_clientからTB2C\_serverへのメタデータ要求

 TB2C\_serverのポートへのHTTP GET

path="/",

param:なし

(6) TB2C\_serverからTB2C\_clientへ返されるメタデータ

 JSON{"id":データID,

　　　"uri":先頭SPHファイルのURI,

　　　"type":"SPH",

　　　"dims":[I,J,K],

　　　"datalen":ベクトル長,

　　　"bbox":[[x0,y0,z0],[x1,y1,z1]],

 　　　"steps":ステップ数,

　　　"timerange":[開始時刻,終了時刻],

　　　"vrange":[最小値,最大値] ,

　　　"vistype":[“isosurf”]}

vistypeには、TB2C\_serverがサポートする可視化図種のリストが入る(今回の実装では“isosurf”のみ)

(7) TB2C\_clientからChOWDERへの接続要求

 ChOWDERへのWebSocket通信

path="ws://chowder\_host/v2",

method:"AddContent"

今回の実装では、chowder\_hostは“localhost”のみ

(8) TB2C\_clientからTB2C\_serverへの図種生成要求

 TB2C ServerのポートへのHTTP PUT

path="/visualize"

param:step=対象ステップ番号,

　　　 vistype="isosurf",

　　　 visparam={"value":等値面の値}

 SPHがベクトルデータの場合は、データの L2ノルムで等値面を生成する

TB2C Serverによる図種生成完了を待ってChOWDERへの描画要求(9)を行う

(9) TB2C\_serverからTB2C\_clientへ返される生成データリスト

JSON[{tiledLayer}, …]

TB2C\_serverで生成された3D-Tilesデータの個数分のtiledLayerのリストが返される

(10) TB2C\_clientからChOWDERへの描画要求

ChOWDERへのWebSocket通信

path="ws://chowder\_host/v2",

method:"UpdateMetaData"

タイムステップまたは等値面の値が変更された場合は、アップデートIDを変更する

視界が変更された場合は、”cameraWorldMatrix”を更新する

(11) ChOWDERからTB2C\_serverへのデータ取得要求

TB2C\_serverのポートへのHTTP GET

path="/visualized/...",

param:なし

(12) TB2C\_serverからChOWDERへのデータリプライ

要求されたファイル

今回の実装ではファイル渡し

# モジュール設計

# Temporal Buffer

(1) TBクラス

class TB(builtins.object)

| TB() -> None

|

| TB - Temporal Buffer

| Temporal Bufferのプロトタイプ実装クラスです。

| JSONまたはファイルリストで指定された時系列SPHデータを読み込み、保持します。

| また、要求されたタイムスライスのデータをエンコードして送信します。

|

| メソッド:

|

| \_\_init\_\_(self) -> None

| コンストラクタ

|

| loadFromFilelist(self, fnlist: [], basedir: str = '.') -> bool

| SPHファイルのリストを時系列データとして読み込みます。

|

| Parameters

| ----------

| fnlist: str[]

| SPHファイルのパスのリスト

| basedir: str

| SPHファイルが存在するディレクトリ(省略時は'.')

|

| Returns

| -------

| bool: True=成功、False=失敗

|

| loadFromJSON(self, json\_path: str) -> bool

| loadFromJSON

| JSONファイルから時系列SPHデータを読み込みます。

| JSONファイルは、以下の形式であることを想定しています。

| {

| "basedir": "SPHファイルが存在するディレクトリ(省略時は'.')"

| "filelist": [

| {"file": "ファイル名1", "step": "タイムステップ番号", "time": "時刻"},

| ...

| ]

| }

| タイムステップ番号と時刻はオプションで、省略された場合SPHファイルに格納されて

| いるタイムステップ番号、時刻が採用されます。

|

| Parameters

| ----------

| json\_path: str

| JSONファイルのパス

|

| Returns

| -------

| bool: True=成功、False=失敗

(2) TBReqHandlerクラス

class TBReqHandler(http.server.SimpleHTTPRequestHandler)

| TBReqHandler(\*args, directory=None, \*\*kwargs)

|

| TBReqHandler

| Temporal Buffer用のHTTPリクエストハンドラー実装クラスです。

|

| メソッド:

|

| do\_GET(self)

| GETメソッド用のリクエストハンドラー

| 要求されたパスが'/'の場合はメタデータを返し、'/quit'の場合は終了します。

| 要求パスが'/data'の場合は、指定されたstepのデータを返します。

(3) TSDataSPHクラス

class TSDataSPH(TSData.TSData)

| TSDataSPH() -> None

|

| TSDataSPH

| 時系列SPHファイル群を扱うクラスです。

|

| メソッド:

|

| \_\_init\_\_(self) -> None

| コンストラクタ

|

| reset(self) -> None

| 初期化

|

| setupFiles(self, fnlist: Iterable, basedir: str = '.') -> bool

| SPHファイルエントリーのリストからクラスパラメータを設定します。

|

| Parameters

| ----------

| fnlist: [{}]

| SPHファイルエントリーのリスト

| basedir: str

| SPHファイルが存在するディレクトリ(省略時は'.')

|

| Returns

| -------

| bool: True=成功、False=失敗

(4) TSDataクラス

class TSData(builtins.object)

| TSData() -> None

|

| TSData

| 時系列データファイル群を扱うクラスです。

|

| メソッド:

|

| \_\_init\_\_(self) -> None

| コンストラクタ

|

| convStepToIdx(self, stp)

| 時系列データのタイムステップ番号からタイムステップインデックス番号に変換します。

|

| Parameters

| ----------

| stp: int

| タイムステップ番号

|

| Returns

| -------

| int: タイムステップインデックス番号

|

| getDataIdx(self, stpIdx)

| stpIdxで指定されたタイムステップインデックス番号のデータを返します。

|

| Parameters

| ----------

| stpIdx: int

| タイムステップインデックス番号

|

| Returns

| -------

| object: データ

|

| getDataStp(self, stp)

| stpIdxで指定されたタイムステップ番号のデータを返します。

|

| Parameters

| ----------

| stp: int

| タイムステップ番号

|

| Returns

| -------

| object: データ

|

| reset(self) -> None

| 初期化

# TB2C server

(1) TB2C\_serverクラス

class TB2C\_server(builtins.object)

| TB2C serverのプロトタイプ実装クラスです。

|

| メソッド:

|

| \_\_init\_\_(self)

| コンストラクタ

|

| connectTB(self, uri: str)

| URIで指定されたデータソースに接続し、メタデータを読み込み、

| 'vistype'を付加して保持します。

|

| Parameters

| ----------

| uri: str

| データソースのURI

|

| generateIsosurf(self, value: float) -> bool

| 現在保持しているSPHデータに対し、valueで指定された値で等値面を生成し、

| 3D-Tiles形式のファイルに出力します。

|

| Parameters

| ----------

| value: float

| 等値面を生成する値

|

| Returns

| -------

| bool: True=成功、False=失敗

|

| getSPHdata(self, id: int, stp: int) -> [<class 'pySPH.SPH.SPH'>]

| TBより、idとstepを指定してSPHデータを取得します。

| 実際にアクセスするURLは'{uri}/data?id={id}&step={stp}'

|

| Parameters

| ----------

| id: int

| 取得するSPHデータのID

| step: int

| 取得するSPHデータのタイムステップインデックス番号

|

| Returns

| -------

| [SPH.SPH]: 取得したデータ(を分割したリスト)

(2) TB2C\_server\_ReqHandlerクラス

class TB2C\_server\_ReqHandler(http.server.SimpleHTTPRequestHandler)

| TB2C\_server\_ReqHandler(\*args, directory=None, \*\*kwargs)

|

| TB2C server用のHTTPリクエストハンドラー実装クラスです。

|

| メソッド:

|

| do\_GET(self)

| GETメソッド用のリクエストハンドラー

| 要求されたパスが'/'の場合はメタデータを返し、'/quit'の場合は終了します。

|

| do\_POST(self)

| POSTメソッド用のリクエストハンドラー

| 要求されたパスが'/visualize'の場合はパラメータに従い可視化を行います。

|

| sendMsgRes(self, code: int, msg: str)

| HTTPアクセスに対するtext/plain形式のレスポンスを返す。

|

| Parameters

| ----------

| code: int

| レスポンスコード

| msg: str

| レスポンスメッセージ

|

| translate\_path(self, path)

| SimpleHTTPRequestHandler.translate\_pathのオーバーロードメソッド。

| SimpleHTTPRequestHandler.do\_GETが呼ばれた際の、リクエストされたパスを

| ファイルシステムのパスに変換する。

| TB2C\_server\_ReqHandlerクラスでは、do\_GETにおいてリクエストパスが

| '/visualized/'で始まる場合のみ(super().do\_GET()から)呼ばれ、

| '/visualized/'までのパスをg\_app.out\_dirに置き換えて返す。

|

| Parameters

| ----------

| path: str

| リクエストパス

|

| Returns

| -------

| str: 変換されたパス

(3) TB2C\_visualizeクラス

class TB2C\_visualize(builtins.object)

| TB2C\_visualize(outdir: str = '.', bbox=[[0, 0, 0], [1, 1, 1]])

|

| TB2Cサーバ用の、可視化機能実装クラスです。

| SPHデータに対する可視化(等値面生成)結果をジオメトリ(OBJ)ファイルに出力し、

| obj23dtilesコマンドを使用して3D-Tilesに変換します。

|

| メソッド:

|

| \_\_init\_\_(self, outdir: str = '.', bbox=[[0, 0, 0], [1, 1, 1]])

| コンストラクタ

|

| bbox2Box(self, bbox: [[<class 'float'>], [<class 'float'>]]) -> [<class 'float'>]

| バウンディングボックスデータを、3D-Tileの"Box"形式に変換します。

|

| Parameters

| ----------

| bbox: [[float],[float]]

| バウンディングボックスデータ

|

| Returns

| -------

| [float]: 3D-Tileの"Box"形式データ

|

| checkB3dmDir(self) -> bool

| 出力先ディレクトリ(self.\_outDir)配下に"b3dm"ディレクトリを

| (存在すれば削除してから)作成します。

|

| Returns

| -------

| bool: True=成功、False=失敗

|

| checkObj23dtiles(self) -> bool

| "obj23dtiles --version"を実行し、正常に動作するか確認します。

|

| Returns

| -------

| bool: True=成功、False=失敗

|

| isosurf(self, sph\_lst: [<class 'pySPH.SPH.SPH'>], value: float, fnbase: str = 'isosurf') -> bool

| sph\_lstで渡されたSPHデータ群に対し、valueで指定された値で等値面を生成し、

| OBJファイルに出力した後、obj23dtilesコマンドを使用して3D-Tilesに変換します。

| self.\_out\_dir配下に、以下のファイルが作成されます。

| b3dm/Batchedfnbase\_nnn/fnbase\_nnn.b3dm

| b3dm/Batchedfnbase\_nnn/tileset.json

|

| Parameters

| ----------

| sph\_lst: [SPH.SPH]

| 等値面を生成するSPHデータのリスト

| value: float

| 等値面を生成する値

| fnbase: str

| 等値面ファイルのベース名(省略時:"isosurf")

|

| Returns

| -------

| bool: True=成功、False=失敗

(4) SPH\_filterクラス

class SPH\_filter(builtins.object)

| スタティックメソッド:

|

| divideShareEdge(d: pySPH.SPH.SPH, div: []) -> []

| SPHデータについて、隣接格子点を共有した分割を行う(static method)

| 格子サイズが5の次元を2分割する場合、分割されたデータの格子サイズは(3, 3)になる.

| 分割されたデータの格子サイズは1以上でならなければならない.

|

| Parameters

| ----------

| d: SPH.SPH

| 分割するSPHデータ

| div: int[3]

| 各軸方向の分割数(>0)

|

| Returns

| -------

| SPH.SPH[]: 分割されたSPHデータのリスト、空のリスト=失敗

|

| extractScalar(d: pySPH.SPH.SPH, dataIdx: int) -> pySPH.SPH.SPH

| ベクトルデータを持つSPHからスカラーのSPHを生成する(static method)

|

| Parameters

| ----------

| d: SPH.SPH

| vectorデータを持つSPHデータ

| dataIdx: int

| 抽出するスカラーデータのインデックス番号

|

| Returns

| -------

| SPH.SPH: 抽出したスカラーのSPHデータ、None: 失敗

|

| fromJSON(sd: str) -> pySPH.SPH.SPH

| base64エンコードでJSON化されたSPHデータを復元する(static method)

|

| Parameters

| ----------

| sd: str

| 文字列化したJSONデータ

|

| Returns

| -------

| SPH.SPH: 復元されたSPHデータ

|

| toJSON(d: pySPH.SPH.SPH) -> str

| SPHデータについて、base64でエンコードしてJSON化する(static method)

|

| Parameters

| ----------

| d: SPH.SPH

| JSON化するSPHデータ

|

| Returns

| -------

| str: 文字列化したJSONデータ

|

| vectorMag(d: pySPH.SPH.SPH) -> pySPH.SPH.SPH

| ベクトルデータを持つSPHからベクトルのノルムをスカラーとして持つSPHを生成する。

| (static method)

|

| Parameters

| ----------

| d: SPH.SPH

| vectorデータを持つSPHデータ

|

| Returns

| -------

| SPH.SPH: ベクトルノルムのスカラーSPHデータ、None: 失敗

# TB2C client

(1) TB2C\_Appクラス

class TB2C\_App(wx.core.App)

| TB2C\_App(redirect=False, filename=None, useBestVisual=False, clearSigInt=True)

|

| TB2C clientのプロトタイプAppクラスです。

| wxPythonのAppクラスを継承しています。

|

| メソッド:

|

| Message(self, msg: str, err: bool = False)

| msgをMessageDialogに表示します。

|

| Parameters

| ----------

| msg: str

| メッセージ文字列

| err: bool

| MessageDialogのアイコンをエラーアイコンにするかどうか

|

| OnClose(self, evt)

| ウインドウクローズ時のイベントハンドラーです。アプリケーションを終了します。

|

| Parameters

| ----------

| evt: wx.Event

| ウインドウクローズイベント

|

| OnConnectChOWDER(self, evt)

| 'Connect to ChOWDDER'メニューのイベントハンドラーです。

| URLおよびパスワード入力用のダイアログを表示し、ChOWDERに接続します。

|

| Parameters

| ----------

| evt: wx.Event

| メニューイベント

|

| OnConnectTB2CSrv(self, evt)

| 'Connect to TB2C server'メニューのイベントハンドラー。

| URL入力用のダイアログを表示し、TB2C serverに接続します。

|

| Parameters

| ----------

| evt: wx.Event

| メニューイベント

|

| OnInit(self)

| App初期化時のイベントハンドラー。

| トップレベルのFrameを作成し、OpenGL canvasとUI panelを横方向に配置します。

|

| OnQuit(self, evt)

| 'Quit'メニューのイベントハンドラーです。アプリケーションを終了します。

|

| Parameters

| ----------

| evt: wx.Event

| メニューイベント

|

| connectChOWDER(self, hostnm: str, pswd: str) -> bool

| hostnmで指定されたホスト上のChOWDER serverに接続します。

|

| Parameters

| ----------

| hostnm: str

| ChOWDER serverが動作するホスト(ホスト名またはIPアドレス)

| 接続先は'ws://{hostnm}/v2'となる。

| pswd: str

| ChOWDER serverに接続する際のAPIUserのパスワード

|

| Returns

| -------

| bool: True=成功、False=失敗(self.\_lastErrにエラーメッセージを登録)

|

| connectTB2CSrv(self, url: str) -> bool

| urlで指定されたTB2C serverへ接続し、メタデータを取得します。

| 取得したメタデータのチェック後、AppのUIに反映します。

|

| Parameters

| ----------

| url: str

| TB2C serverのURL

|

| Returns

| -------

| bool: True=成功、False=失敗(self.\_lastErrにエラーメッセージを登録)

|

| updateRequest(self, flag) -> bool

| updateRequest

| データ更新処理を行います。

| flagのTB2C\_App.REQ\_UPDDATAビットがONの場合は、TB2Cサーバに等値面の再作成を

| 依頼し、ChOWDERに表示更新を依頼します。これは、タイムステップまたは等値面の値が

| 変更された場合にコールされます。

| flagのTB2C\_App.REQ\_UPDVIEWビットがONの場合はChOWDERへの表示更新依頼のみ

| を行います。これは視界が変更された場合にコールされます。

|

| Parameters

| ----------

| flag: int

| 更新要求フラグ

|

| Returns

| -------

| bool: True=成功、False=失敗(self.\_lastErrにエラーメッセージを登録)

(2) TB2C\_Canvasクラス

class TB2C\_Canvas(wx.\_glcanvas.GLCanvas)

| TB2C\_Canvas(parent, app)

|

| TB2C clientのOpenGL表示用キャンバスクラスです。

| wxPythonのGLCanvasクラスを継承しており、マウスイベント用のハンドラが実装されています。

|

| Draw(self)

| 描画処理を行います。

|

| GetFitMatrix(self)

| オブジェクトを視界にフィットさせる変換行列を返ます。

|

| Returns

| -------

| Mat4: 変換行列

|

| GetMatrix(self)

| カメラの変換行列を返ます。

|

| Returns

| -------

| Mat4: カメラ変換行列

|

| OnDoubleClick(self, evt)

| ダブルクリックに対するイベントハンドラーです。視界のリセットを行います。

|

| Parameters

| ----------

| evt: wx.Event

| マウスイベント

|

| OnMouseDown(self, evt)

| マウスボタン押下に対するイベントハンドラーです。

|

| Parameters

| ----------

| evt: wx.Event

| マウスイベント

|

| OnMouseMotion(self, evt)

| マウス移動に対するイベントハンドラーです。

|

| Parameters

| ----------

| evt: wx.Event

| マウスイベント

|

| OnMouseUp(self, evt)

| マウスボタンリリースに対するイベントハンドラーです。

|

| Parameters

| ----------

| evt: wx.Event

| マウスイベント

|

| OnMouseWheel(self, evt)

| マウスホイール回転に対するイベントハンドラーです。

|

| Parameters

| ----------

| evt: wx.Event

| マウスイベント

|

| OnPaint(self, event)

| 再描画に対するイベントハンドラーです。描画処理を行います。

|

| Parameters

| ----------

| evt: wx.Event

| 再描画イベント

|

| OnSize(self, event)

| キャンバスサイズの変更に対するイベントハンドラーです。ビューポート変更を行います。

|

| Parameters

| ----------

| evt: wx.Event

| サイズイベント

|

| setBoxSize(self, minpos, maxpos)

| 表示オブジェクトのバウンディングボックスサイズを設定します。

|

| Parameters

| ----------

| evtminpos: [float]

| 最小座標値

| evtmaxpos: [float]

| 最大座標値

(3) TB2C\_UIPanelクラス

class TB2C\_UIPanel(wx.\_core.Panel)

| TB2C\_UIPanel(parent, app, size=wx.Size(250, -1))

|

| メソッド:

|

| \_\_init\_\_(self, parent, app, size=wx.Size(250, -1))

| コンストラクタ

|

| OnIsovalSlider(self, evt)

| 等値面の値スライダー操作に対するイベントハンドラーです。

|

| Parameters

| ----------

| evt: wx.Event

| スライダーイベント

|

| OnIsovalTxt(self, evt)

| 等値面の値テキストボックス入力に対するイベントハンドラーです。

|

| Parameters

| ----------

| evt: wx.Event

| テキストボックス入力イベント

|

| OnTsSlider(self, evt)

| タイムステップスライダー操作に対するイベントハンドラーです。

|

| Parameters

| ----------

| evt: wx.Event

| スライダーイベント

|

| OnTsTxt(self, evt)

| タイムステップテキストボックス入力に対するイベントハンドラーです。

|

| Parameters

| ----------

| evt: wx.Event

| テキストボックス入力イベント

|

| setInformation(self, info: str)

| 情報表示欄の表示内容設定。

|

| Parameters

| ----------

| info: str

| 表示内容

|

| setTimeStepRange(self, steps: int) -> bool

| タイムステップ数の設定。

|

| Parameters

| ----------

| steps: int

| タイムステップ数

|

| setValueRange(self, vrange: []) -> bool

| データ値域の設定。

|

| Parameters

| ----------

| vrange: [float]

| データ値域

(4) ConnectChOWDERDlgクラス

class ConnectChOWDERDlg(wx.\_core.Dialog)

| ConnectChOWDERDlg(hostname: str = None)

|

| ChOWDER接続用のダイアログクラスです。

| ChOWDERホスト名とパスワードの入力テキストボックスを配置しています。

(5) ConnectTB2CSrvDlgクラス

class ConnectTB2CSrvDlg(wx.\_core.Dialog)

| ConnectTB2CSrvDlg(url: str = None)

|

| TB2C server接続用のダイアログクラスです。

| TB2C serverの接続URLの入力テキストボックスを配置しています。

(6) Frustumクラス

class Frustum(builtins.object)

| 視垂台クラス

| 視垂台はモデル空間の中の視界を表わす垂台の領域です。

| eye: 視点座標

| view: 視線方向ベクトル

| up: 上方向ベクトル

| dist: 視点から注視点までの距離

| halfW/halfH: 注視点における視界の幅および高さの半分の値

| near/far: 視点から前後のクリップ面までの距離

|

| メソッド:

|

| \_\_init\_\_(self)

| コンストラクタ

|

| ApplyModelview(self)

| モデルビュー行列のOpenGL適用

| 以下の変換を行う行列を生成し、モデルビュー行列とする

| - ステレオ表示の場合、視点位置をオフセットする

| - 視線方向の回転

| - 視点位置への平行移動

|

| ApplyProjection(self, ortho=False, asp=1.0)

| プロジェクション行列のOpenGL適用

|

| Parameters

| ----------

| ortho: bool

| 平行投影モード

| asp: float

| 視界のアスペクト比率(横/縦)

|

| GetChOWDERMatrix(self)

| ChOWDER用のカメラ変換行列を返す

|

| Returns

| -------

| Mat4: ChOWDER用のカメラ変換行列

|

| GetMVM(self)

| モデルビュー行列を返す

|

| Returns

| -------

| Mat4: モデルビュー行列

|

| GetPM(self, ortho=False, asp=1.0)

| プロジェクション行列を返す

|

| Parameters

| ----------

| ortho: bool

| 平行投影モード

| asp: float

| 視界のアスペクト比率(横/縦)

|

| resetEye(self)

| 視界をリセットする。

|

| rotHead(self, a)

| up周りにviewをa(deg)回転させる

|

| Parameters

| ----------

| a: float

| 回転角度

|

| rotPan(self, a)

| Right(=up x (-view))周りにview, upをa(deg)回転させる

|

| Parameters

| ----------

| a: float

| 回転角度

|

| trans(self, x, y, z)

| eyeをRight(=up x (-view))方向にx, up方向にy, view方向にz移動させる

|

| Parameters

| ----------

| x: float

| Right方向移動量

| y: float

| Up方向移動量

| z: float

| view方向移動量

# 動作検証報告

# 概要

「富岳」を想定したHPCI計算資源を用いての動作検証として、九州大学ITOシステムの占有フロントエンドノード(ITOフロントエンド)にTB2CシステムおよびChOWDERシステムをインストールし、ここで動作するサーバー群とローカルPC上で動作するTB2Cクライアントを接続して動作検証を行います。

この際に、ChOWDERディスプレイとして、理化学研究所R-CCSのタイルドディスプレイ装置に表示するWebブラウザを使用し、ChOWDERのレンダリング時間計測機能を使用してレンダリング性能を測定します。

なお、ITOフロントエンドに割り当てられるIPアドレスはプライベートアドレスであり、ITOログインノードを介したssh接続しか行うことができないため、動作検証作業に際してはITOログインノードへの通信をsshのポートフォワーディング機能を使用してITOフロントエンドに転送する設定を行います。

また、ITOフロントエンドではroot権限が与えられないため、TB2CシステムおよびChOWDERシステムのインストールに関わる全てのソフトウエアのインストールはユーザー環境に行います。

ChOWDERが使用する通信ポートは、デフォルトでは80番(および443番)ですが、今回の作業ではroot権限を必要としない1024番以上のポート(8080番)を使用します。

# ITOフロントエンドでの環境構築

# ノード構成

* ノードテンプレート：BLGI(ベアメタル)
* OS：Red Hat Enterprise Linux Server release 7.3
* CPU：36コア、主記憶：384GiB

# Spackのインストールおよび設定

ITOフロントエンドでPython3、Node.jsおよびこれらの関連モジュールをユーザー環境にインストールするため、パッケージ管理システムSpackを使用します。

以下のコマンドをITOフロントエンド上で実行することで、Spackがインストールされます。

　cd $HOME

　git clone https://github.com/spack/spack.git

これにより、ホームディレクトリ配下にspackというディレクトリが作成され、ここにSpackの実行環境がダウンロードされます。ここで、以下のコマンドを実行するとspackコマンドが利用可能になります。

　source $HOME/spack/share/spack/setup-env.sh

ITOフロントエンドでは、environment-moduleシステムで複数のコンパイラが使用可能となっていますが、今回の作業ではgcc-9.2.0を使用します。

以下のコマンドをITOフロントエンド上で実行することで、Spackがコンパイラとしてgcc-9.2.0を使用可能になります。

　module load gcc/9.2.0

　spack compiler find

上記のコマンドを実行することにより、$HOME/.spack/linux/compiler.yamlファイルが作成され、Spackのコンパイラーリストにgcc@9.2.0が追加されます。

最後に、$HOME/.spack/packages.yamlファイルを作成し、以下の内容を記述します。

packages:

perl:

externals:

- spec: "perl@5.32.0"

prefix: /usr

buildable: False

all:

compiler: ['gcc@9.2.0', 'gcc@4.8.5', 'gcc@4.4.7']

target: [x86\_64]

これにより、Spackは/usrにインストールされているPerlを使用し、パッケージとしてインストールすることはなくなります。

# Node.jsおよびnpmのインストール

Spackを使用してNode.jsおよびnpmのインストールを行うには、以下のコマンドを実行します。

　spack install node-js%gcc@9.2.0

　spack install npm%gcc@9.2.0

インストール終了後は、以下のコマンドを実行するとnodeおよびnpmコマンドが利用可能になります。

　spack load node-js%gcc@9.2.0

　spack load npm%gcc@9.2.0

次に、インストールしたnpmを使用してobj23dtilesをインストールします。以下のコマンドを実行します。

　npm install -g obj23dtiles

# Python3およびPythonモジュール群のインストール

Spackを使用してPython3のインストールを行うには、以下のコマンドを実行します。

　spack install python%gcc@9.2.0

インストール終了後は、以下のコマンドを実行するとpython3およびpip3コマンドが利用可能になります。

　spack load python%gcc@9.2.0

次に、インストールしたPython3(pip3コマンド)を使用して、TB2Cの実行に必要なPythonモジュール群をインストールします。以下のコマンドを実行します。

　pip3 install scikit-image

　pip3 install wxPython

　pip3 install pyOpenGL

　pip3 install websocket-client

# Spack環境のbash設定

Spackを使用してインストールした環境を、次回ログイン時にも再現できるように、$HOME/.bash\_profileに以下の記述を追加しておきます。

if [ -d $HOME/spack ]; then

source $HOME/spack/share/spack/setup-env.sh

spack load python%gcc@9.2.0

spack load node-js%gcc@9.2.0

spack load npm%gcc@9.2.0

fi

# ChOWDERのインストール

ChOWDERのインストールは、ChOWDERのgithubリポジトリよりソースをダウンロード(clone)し、インストールスクリプトを実行することで行います。

まず、ソースのダウンロードを行います。以下のコマンドを実行します。

　cd $HOME

　git clone -b 202009 http://github.com/digirea/ChOWDER.git

これにより、ホームディレクトリ配下のChOWDERディレクトリに、ChOWDERのソース一式(202009ブランチ)がダウンロードされます。

次に、インストールスクリプトを実行します。以下のコマンドを実行します。

　cd $HOME/ChOWDER/bin

　sh ./install.sh

通常、インストールスクリプトの実行でChOWDERのインストールは完了しますが、今回使用しているChOWDERのブランチ(202009ブランチ)では、CentOS 7用のRedisサーバーがインストールされません。そこで、以下のようにRedisのソースコードをダウンロードし、Redisサーバーのインストールを行います。

　wget http://download.redis.io/releases/redis-5.0.5.tar.gz

　tar xvfz redis-5.0.5.tar.gz

　cd redis-5.0.5

　make

コンパイルが終了したら、Redisサーバーの実行ファイルをChOWDER環境下にコピーします。

　cp src/redis-server $HOME/ChOWDER/redis/

# TB2Cのインストール

TB2Cのインストールは、TB2Cの提供ファイル(TB2C-1.x.tar.gz)をsftpでITOフロントエンドにコピーし、任意のディレクトリで展開することで行えます。

ここでは、ホームディレクトリ配下のTB2Cディレクトリ以下に展開します。

　cd $HOME

　tar xvfz TB2C-1.x.tar.gz

# 実施方法

* + 1. **ITOフロントエンドでのサーバー実行**

ITOフロントエンド上で、Tenporal Buffer、TB2C serverおよびChOWDERを動作させておく必要があります。

尚、以下の説明ではTB2C展開ディレクトリは$HOME/TB2C、ChOWDER展開ディレクトリは$HOME/ChOWDERとしています。

(1) Temporal Buffer

ITOフロントエンド上のTB2C展開ディレクトリに移動し、以下のコマンドを実行します。

　cd $HOME/TB2C

　python3 python/TB.py -j data/concat\_input\_p.json

(2) TB2C server

ITOフロントエンド上のTB2C展開ディレクトリに移動し、以下のコマンドを実行します。

　cd $HOME/TB2C

　python3 python/TB2C\_server.py --odir $HOME/ChOWDER/public/data ¥

　　　--dx 2 --dy 2

(3) ChOWDER

ITOフロントエンド上のChOWDER展開ディレクトリ/bin に移動し、以下のコマンドを実行します。

　cd $HOME/ChOWDER/bin

　sh ./run.sh

# ITOログインノードでのポートフォワーディング設定

ITOログインノード(ito.cc.kyushu-u.ac.jp)上では、外部からの8080番ポートへの接続をITOフロントエンドの8080番ポートに、また外部からの4000番ポートへの接続をITOフロントエンドの4000番ポートに、それぞれポートフォワーディングする必要があります。

これには、ITOログインノード上で以下のsshコマンドを実行します。ここで、xx.xx.xx.xx にはITOフロントエンドノードのIPアドレスを指定します。

■ 8080番ポートのポートフォワーディング

　ssh -g -L 8080:xx.xx.xx.xx:8080 xx.xx.xx.xx

■ 4000番ポートのポートフォワーディング

　ssh -g -L 4000:xx.xx.xx.xx:4000 xx.xx.xx.xx

これらは、ITOログインノードに別個にログインした別のターミナルで実行する必要があります。

また、ITOログインノードはito-1とito-2の２台のマシンが負荷分散のために切り替えられて使用されているため(本書執筆時点)、ito-1とito-2の両方のマシン上で上記のコマンドを実行する必要があります。

# ローカルPCでのポートフォワーディング設定

ローカルPC(TB2C clientを実行するマシン)上では、80番ポートへの接続をITOログインノードの8080番ポートに、また4000番ポートへの接続をITOグインノードの4000番ポートに、それぞれポートフォワーディングする必要があります。

これには、ローカルPC上で以下のsshコマンドを実行します。

■ 80番ポートのポートフォワーディング

　ssh -i 秘密鍵 -L 80:ito.cc.kyushu-u.ac.jp:8080 ユーザー名@ito.cc.kyushu-u.ac.jp

■ 4000番ポートのポートフォワーディング

　ssh -i 秘密鍵 -L 4000:ito.cc.kyushu-u.ac.jp:4000 ユーザー名@ito.cc.kyushu-u.ac.jp

これらは、ローカルPC上の別個のターミナルで実行する必要があります。

# ChOWDERコントローラーの実行

ローカルPC上でWebブラウザを起動し、以下のURLに接続することでChOWDERコントローラーが表示されます。

　http://localhost:80/

80番ポートのポートフォワーディングがされているので、ITOフロントエンド上のChOWDERサーバーに接続しています。

# TB2C clientの実行

ローカルPC上でTB2C clientを実行し、ITOフロントエンド上のTB2C serverおよびChOWDERサーバーに接続させます。ローカルPC上のTB2C展開ディレクトリに移動し、以下のコマンドを実行します。

　python3 python/TB2C\_client.py -s http://localhost:4000/ -c localhost

上記コマンドを実行すると、TB2C clientのウインドウが表示され、ChOWDERのAPIUserのパスワード入力が求められます。パスワードを入力すると、ChOWDERコントローラーにはContent IDがtb2c\_3dtileであるコンテンツが登録されます。

ここで、TB2C clientのGUIでtimestep indexを40に、isosurf valueを0.0に設定すると、下図のような表示になります。



# ChOWDERディスプレイのレンダリング時間測定

ChOWDERディスプレイの接続を行い、ローカルPC上のTB2C展開ディレクトリに移動して、以下のコマンドを実行すると、ChOWDERディスプレイのレンダリング時間の測定が行われます。

　python3 python/chowder\_measure.py

# 実施結果

* + 1. **ITOフロントエンド・R-CCSタイルドディスプレイでの実行結果**

ITOフロントエンド上でTenporal Buffer、TB2C serverおよびChOWDERサーバーを動作させ、ChOWDERディスプレイを理化学研究所R-CCSのタイルドディスプレイ装置で動作させた場合(4画面)のレンダリング時間測定結果を以下に示します。

Measure Result: {'nodeVisible': {}, 'textureCount': 0, 'geometryCount': 13,  
 'triangleCount': 6722, 'pointCount': 0, 'lineCount': 25, 'updateDuration': 0}

Measure Result: {'nodeVisible': {}, 'textureCount': 0, 'geometryCount': 13,  
 'triangleCount': 0, 'pointCount': 0, 'lineCount': 25, 'updateDuration': 0}

Measure Result: {'nodeVisible': {}, 'textureCount': 0, 'geometryCount': 13,  
 'triangleCount': 30650, 'pointCount': 0, 'lineCount': 25, 'updateDuration': 1}

Measure Result: {'nodeVisible': {}, 'textureCount': 0, 'geometryCount': 13,  
 'triangleCount': 0, 'pointCount': 0, 'lineCount': 25, 'updateDuration': 0}

* + 1. **課題**

今回の作業において明らかになった課題および対応策について、以下に記述します。

(1) SSHポートフォワーディング設定の煩雑さ

今回の作業で動作検証を行った環境は、サーバーソフトウエア群を動作させるマシンのIPアドレスがプライベートアドレスであるため、SSHによる多段ポートフォワーディングを行う必要がありました。

本システムではソフトウエア間の通信を行うポートが複数あり、ポートフォワーディングの設定も各ポート毎に行う必要があるため、その設定作業は非常に煩雑となりました。

この課題に対しては、ポートフォワーディングの設定を自動または半自動で行うスクリプトの作成が有効であると考えられます。

(2) システムソフトウエアインストールの手間の大きさ

今回の作業で動作検証を行った環境では、サーバーソフトウエア群を動作させるマシンのroot権限が得られないため、システムを動作させるために必要なシステムソフトウエアを全てユーザー環境にインストールする必要がありました。

今回の作業では、この課題への対応としてSpackの利用を行い、有効な対応策であることが分かりましたが、この他の解決策としてソフトウエア一式がインストールされた仮想環境(Docker, Singularity)を作成し、配布することが考えられます。

(3) TB2C serverの並列化未対応

今回の作業で使用したデータは、テスト用に用意された比較的小規模なものであるため、TB2Cシステムの動作におけるパフォーマンスの問題は、それほど深刻なものではありませんでした。

しかし、実際の運用シーンにおいて大規模なデータを処理する場合、TB2Cがシリアル実行(非並列実行)で動作していることは、深刻なパフォーマンス低下を惹起することが容易に想像されます。

この問題に対する解決策は、TB2Cシステムの並列化です。特に、TB2C serverの処理は並列化が有効であると考えられます。一方、Temporal Bufferの処理の並列化は、外部インターフェースをどのように構成するかを併せて考える必要があり、検討が必要だと考えられます。

以上