CORBA オペレーション呼び出しの高速化と RT ミドルウエアへの適用

○安藤慶昭 (産総研)

1. はじめに

著者らが開発する RT ミドルウエア: OpenRTM-aist[1] は、分散ミドルウエアとして CORBA を利用している。CORBA は従来、銀行や軍の指揮・統制等のエンタープライズ系システム、通信インフラ系など大規模システムにおいて利用されてきたが、近年の計算機性能の向上から、組込み系、制御系にも利用されつつある [2]。しかしながら、高周期のリアルタイム実行などでは、僅かなオーバーヘッドが問題となるケースも依然存在する。本稿では、RT コンポーネントをリアルタイム実行する際に顕在化する可能性のある CORBA のオーバーヘッドについて検証し、これを回避する方法を提案する。

2. CORBA

CORBA (Common Object Request Broker Architecture) は OMG で標準化されている分散オブジェクトの標準仕様である [3]。CORBA の基本的な構造を図 1 に示す。

IDL (Interface Definition Language) によりインターフェースを定義し、通常 IDL コンパイラから**スタブ**と**スケルトン**を自動生成する。機能を提供するサーバオブジェクトは、スケルトンを継承するなどして実装する。これをインスタンス化したものを**サーバント**と呼ぶ。一方クライアント側は、サーバ側で生成されたオブジェクトの参照 (**オブジェクト参照**と呼ぶ) を何らかの方法で取得し、オブジェクト参照に対して、オペレーション (メンバ関数, メソッド) を呼び出す。呼び出しに関する情報は言語に依存しない CDR (Common Data Representation) と呼ばれる共通の方法で直列化 (マーシャリング) され、サーバ側に伝達される。

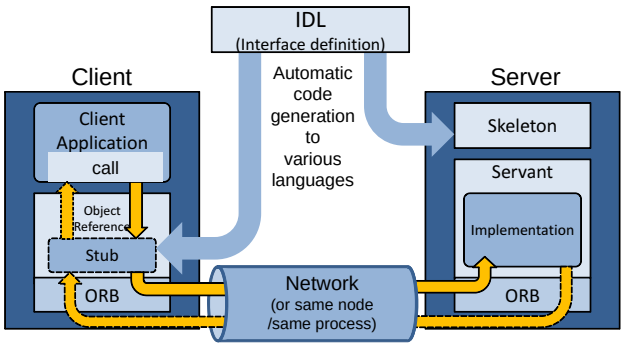

図 1 CORBA (Common Object Request Broker Architecture).

2·1 ローカルコールは十分に速いか？

CORBA 標準仕様では実装方法について規定していないが、多くの CORBA 実装では効率化のために、クライアントとサーバのメモリ空間が同一であるローカルオブジェクトへの呼び出しは、マーシャリング等を省いて単なる関数コールで行なっている。従って、一般には「ローカルオブジェクト呼び出しは十分早い」と言われている。これを定量的に評価するために、同一プロセス内の CORBA オブジェクトへのオブジェクト参照を介した呼び出しと、サーバントへの直接呼び出しの比較実験を行った。

```
interface Hoge
{
  void munya(in string msg, in short val);
};
```

図 2 実験に用いたインターフェース定義.

上記のオペレーションを一つ持つインターフェースを定義し、これを C++で実装した。関数 `munya()` は実際には何もしない関数である。

```
Hoge_impl* hoge_servant = new Hoge_impl();
CORBA::Object_var hoge_obj = hoge_servant->_this();
Hoge_var hoge_var = Hoge::_narrow(hoge_obj);
for (int i(0); i < 1000000; ++i) {
  hoge_var->munya("hoge", 1);     // 1) Object ref.
}
for (int i(0); i < 1000000; ++i) {
  hoge_servant->munya("hoge", 1); // 2) Servant
}
```

図 3 サーバント呼び出しとオブジェクト参照呼び出し.

図 3 に示すように、サーバント (`hoge_servant`) とオブジェクト参照 `hoge_var` への呼び出しをそれぞれ 1000000 回づつ行い、呼び出し 1 回あたりに要した平均時間を計測した結果を表 1 に示す。サーバントの直接呼び出しに対して、オブジェクト参照への呼び出しは 42 倍の時間がかかっていることがわかる。

表 1 サーバントとオブジェクト参照呼び出し時間の比較.

Target Object	Time [ns/call]
1) Object reference	150.9
2) Servant	3.571

クライアントとサーバが疎結合であり、呼び出し周期 (または頻度) が呼び出しオーバーヘッド (上の例では 150 ns) よりも十分に長い場合は無視できる。同一

環境、同等のオブジェクトの呼び出しを Python で行った場合では 268 μs/call であったことを考慮すると、数十 μs オーダーの応答で十分機能する処理に対しては、「十分に速い」と言える。

2·2 RTC の直列実行時の問題

RT コンポーネント (RTC: RT-Component) のコアロジックは、実行コンテキスト (EC: Execution Context) と呼ばれる抽象化されたスレッドオブジェクトからのコールバック呼び出しにより駆動される。この場合、EC はクライアントであり、RTC がサーバントとなる。RTC がアクティブ状態の時、周期実行用のコールバック `onExecute()` 関数が EC から一定周期で呼び出される。

RTC と EC の関係は多対多の関連付けが可能であり、1 つの EC に対して複数の RTC を関連付けることで、RTC を直列的に実行することが可能である。EC をリアルタイム実行可能なものに交換すれば、RTC 内部のロジックを書き換えることなくサーボ等リアリタイム処理に利用することができる。

実際、EC 内部では図 4 のように、複数の RTC オブジェクトのコールバック関数を順次呼び出している。

```
  :
for (int i(0); i < rtcs.size(); ++i) {
  rtcs[i]->onExecute(ec_id); // CORBA obj invocation
}
  :
```

図 4 EC 内での RTC の呼び出し (擬似コード).

この時、オブジェクト参照に対する呼び出し時間を t_{objref}, 直列に実行される RTC の数を N_{RTC} とすると、全体のオーバーヘッド T は

$$T = t_{objref} \times N_{RTC},$$

となる。仮に、10 個のコンポーネントを直列実行した場合、上述の実験結果を踏まえると、

$$T = 151[\text{ns}] \times 10 = 1.51[\mu\text{s}]$$

オーバーヘッドは 1.5 μs となる。実行周期に対するオーバーヘッドの割合は、実行周期が 1 [ms] であれば 1.5%、100[μs] であれば 15% にも達する。

3. 高速化手法

CORBA 標準仕様の POA (Portable Object Adapter) クラスには、`reference_to_servant()` と呼ばれる関数が定義されている。

```
Servant  PortableServer::POA::
     reference_to_servant(in Object reference);
```

図 5 擬似 IDL による reference_to_servant() 関数宣言.

これは、ORB が管理するオブジェクトテーブルを参照して、当該オブジェクト参照に対応するサーバントへのポインタを返す関数である。サーバントが同一メモリ空間上にないリモートオブジェクトに対してこの関数を呼び出すと NULL が返る。

これを利用することで、オブジェクト参照への呼び出しをサーバントへの直接の呼び出しに変換し、高速化を図ることができると考えた。

4. RT ミドルウエアへの適用

アタッチされた RTC のオブジェクト参照を受け取る EC の関数内で、`reference_to_servant()` 関数でサーバントの取得を試み、取得できた場合サーバントのポインタを保持するよう OpenRTM-aist のコードを変更した。

```
void RTObjectStateMachine::
setComponentAction(const RTObject_ptr comp) {
    :
  m_caVar = RTC::ComponentAction::_narrow(comp);
  m_rtobjPtr = dynamic_cast<RTC::RTObject_impl*>
        (poa->reference_to_servant(comp));
    :
```

図 6 EC の修正部分.

RTC の各コールバックを呼び出す部分では、`m_rtobjPtr` が NULL でなければサーバントの、NULL であればオブジェクト参照のそれぞれオペレーションを呼び出すよう変更する。

4·1 実験

OpenRTM-aist 最新版 (trunk, r2355), omniORB-4.1.2 を利用し、従来の EC とサーバント呼び出しを行う EC の呼び出し時間の比較を、空の RTC1 個を EC から 1000000 回呼び出し平均値を計測することで行った。

表 2 EC から RTC 呼び出し時間の比較.

Target Object	Time [ns/call]
1) Object reference	644.4
2) Servant	65.07

表 2 に示すように、サーバント呼び出しはオブジェクト参照呼び出しの約 10 倍高速であり、仮に 1ms 周期で RTC を 10 個直列実行した場合も、オーバーヘッドは周期の 0.65%程度である。

5. おわりに

本稿では、RT ミドルウエアで用いられる CORBA のオブジェクト参照呼び出しに起因するオーバーヘッドについて解析し、より高速に呼び出し可能なサーバント呼び出し方法を実行コンテキストに適用した。この方法は、データポート等繰り返し呼び出しが行われる部分においても有効であり、今後はこうした部分への適用を行なっていく。

参考文献

[1] “OpenRTM-aist”, `http://openrtm.org`
[2] Douglas C. Schmidt and Fred Kuhns. ”An Overview of the Real-time CORBA Specification,” IEEE Computer Magazine, June 2000
[3] “CORBA Specification, Version 3.2”, OMG specification formal/2011-11-01∼03