

## 海量数据分页机制在 Web 信息系统中的应用研究

勾成图, 张 璟, 李军怀

(西安理工大学 计算机科学与工程学院, 陕西 西安 710048)

(zhangjing@xaut.edu.cn)

**摘 要:**在 Web 开发中,应用微软 VS. NET 集成开发环境中提供的 DataGrid 等数据控件,采用传统分页技术,很难解决海量数据的传输,分页的快速查询。结合一大型企业 Web 信息系统的开发实践,提出了基于 XML 和基于存储过程的分页算法,并结合实例对这些算法进行了详细的阐述。测试表明,其性能与响应速度相对传统分页技术有了很大的提高。

**关键词:**分页; Web; XML; 存储过程

**中图分类号:** TP311 **文献标识码:** A

## Application and study of massive-scale data paging mechanism in Web information system

GOU Cheng-tu, ZHANG Jing, LI Jun-huai

(School of Computer Science and Engineering, Xi'an University of Technology, Xi'an Shaanxi 710048, China)

**Abstract:** In Web development, It's difficult to transmit massive-scale data and to query in paging quickly by applying the Data Controls such as DataGrid of Microsoft VS. NET IDE and using the traditional paging-technology. In the development of someone enterprise Web Information system, several paging algorithms based on XML and storage process were proposed. Detailed description of these paging algorithms were given combined with an application instance. The tests manifest that the performance and executing speed are enhanced much in contrast with traditional paging technology.

**Key words:** paging; Web; XML; storage process

### 0 引言

在企业级 Web 信息系统中,用户浏览检索的信息可能是大量甚至海量的,而用户每次在页面上查看的数据信息不过是其中的一部分。这就需要将结果数据以分页的方式显示在 Web 页面上,继而逐页浏览。在诸如微软的 VS. NET 集成开发工具中虽然提供了类似 DataGrid 这样功能强大、使用方便的分页控件,其常规的分页方法难以适应针对大数据量的分页。因为常规分页方法需开启控件的 ViewState 功能才能保存这些控件的数据状态,这势必要消耗大量的内存资源。在页面提交或转向另一页面时,DataGrid 和 DataSet 这些对象都因被销毁而需重建,这意味着全部结果集将频繁在数据库服务器、Web 服务器和客户端之间传输。数据量大时使用 Session 保存或使用 Cache 来缓存对象亦显得不合适,其瓶颈在于 Web 服务器承受的压力过大,内存资源等被长时间占用,利用率低。

为了解决前述常规分页机制的缺陷,我们提出了两种解决办法,分别是基于 XML 的分页机制和基于存储过程的分页机制。经过我们测试,这两种分页方法在对海量数据进行分页查询时的执行效率很高。

### 1 基于 XML 的分页机制

#### 1.1 基于 XML 分页思想

基于 XML 分页思想是只访问一次底层数据库而把符合

条件的数据全部读取至 Web 服务器,并在其上生成一个 XML 文件,使得客户端直接和 XML 文件进行交互。当存在多用户在同一时间段对相同数据源频繁查询时,这些大量的相同数据反复在网络中传输,无疑是对宝贵的数据库服务器资源和带宽的极大浪费。加之考虑到在企业多个系统集成中,某些数据源本身就是 XML 文件,这就要求我们对 XML 文件进行条件查询并分页显示查询结果。采用基于 XML 的分页方法可避免相同数据在服务器间频繁交互传输,这种方法特别适用于对查询频繁且变动性不大的数据(例如对历史数据汇总的查询)。

对于由数据库生成 XML 文件可从两方面考虑。从数据生成时间考虑,XML 文件可由首位查询用户生成,当有新的查询用户时,经过判断,发现其结果数据的 XML 文件存在,便可直接从其中提取数据。XML 文件也可由数据录入人员(具有录入、修改数据权限)在录入或修改数据后生成。这种做法可以省去查询用户生成 XML 的时间,从实际角度考虑,查询用户的职能级别(例如领导)往往很高,更关注响应时间。从 XML 数据量考虑,例如用户要查询 50 车间 5 月份的数据。我们可将 50 车间全年的数据生成 XML 文件。当有户查询 50 车间 6 月份数据时,可借助 XPath 或 XQuery 等 XML 查询语言从全年数据的 XML 文件中直接提取 6 月份数据。

#### 1.2 基于 XML 的分页算法

基于 XML 的分页算法描述如下:

1) 连接访问数据库,从数据库中获取要查询的结果装载在 DataSet 中;

收稿日期:2005-01-13;修订日期:2005-03-19

基金项目:国家 863 计划资助项目(2003AA122560);陕西省教育厅产业化培育项目(01ZC24,02JC43)

作者简介:勾成图(1980-),男,河南新乡人,硕士研究生,主要研究方向:Web 技术及应用;张璟(1952-),男,陕西宝鸡人,教授,博士生导师,主要研究方向:Internet 技术及其应用;李军怀(1969-),男,陕西宝鸡人,副教授,研究方向:数据网格技术、分布式系统、计算机网络应用。

2) 利用 XmlDocument 将 DataSet 中的结果集以 XML 形式保存在 Web 服务器一个指定目录下,并为生成的 XML 文件命名,同时断开数据库连接;

3) 如果有用户访问,首先判断是否在特定的目录下有符合条件的 XML 文件,如果有则不用从数据库中获取,如果没有则转 1);

4) 创建 XmlDocument 对象,并用其 Load 方法加载 XML 文件;

5) 利用 XPath 或者 XQuery 查询技术,对 XML 进行查询,生成自己需要的结果集;

6) 在内存中建立 DataTable, DataTable 的列由要查询的字段名确定;

7) 遍历 5) 中生成的结果集,从结果集中提取要显示一页的记录填入 DataTable 中;

8) 将 DataTable 绑定到 DataGrid, DataGrid 显示该页面的数据。

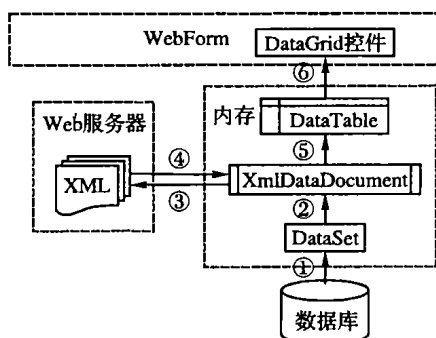


图1 基于XML的分页机制流程

## 2 基于存储过程的分页机制

鉴于数据库的运算能力和稳定性远远高于 Web 服务器,且数据库本身也具有 cache,可利用其反复执行存储过程时速度快和效率高的特点来弥补反复连接访问数据库的不足。我们实现了两种基于存储过程的分页算法。通过这两种算法还可派生出功能更为强大灵活的其他分页方法,其大致流程如图2所示。

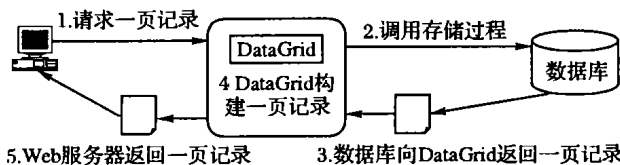


图2 基于存储过程的分页机制流程

这里的算法思想对 SQL Server 和 Oracle 都是适用的。为方便起见,给出为描述该分页算法的一些定义和假设。

**定义** OrderByID(X)操作为对 X 结果集按照 ID 字段进行的排序操作。

设 T 为要查询的总的记录集合,其排序字段为 ID; x 为 T 中任意记录项,即  $x \in T$ 。

E 为最终符合条件的一页有序结果记录集合(即我们的最终结果),有  $E \subseteq T$ ;

E' 为一个与 E 包含相同元素的集合,但其元素没有按 ID 升序排列,  $E = \text{OrderByID}(E')$ 。

### 2.1 基于存储过程的表变量分页机制

这个方法的思想是,根据要返回的结果在存储过程中生成一个 table 变量,另外增加整数类型列 RowNum,指定其为自增长的标识符列。当该表插入数据时,这个列自动增加,起排序作用。另外有两个输入参数 @ StartRow 和 @ StopRow,分别代表要返回的开始记录和结束记录在临时表中的位置,即

要显示在页面上的第一条记录标识的值和最后一条记录标识的值。我们利用 SQL 语句 Set RowCount @ StopRow 和 Insert 来控制向该临时表中插入查询后的记录数。例如要在一个每页显示 100 条记录的记录集中返回第 3 页,则设置 @ StartRow 为 201, @ StopRow 为 300,无需插入大于 300 条记录的数据。最后从临时表中选择 RowNum > = @ StartRow 的记录即是最终结果。table 变量存储在内存中,在存储过程结束后自动释放。当然也可以采用别的建立临时表的方法来实现,例如在 SQL Server 中的 tempdb 数据库中建立的临时表 #Table 等。

采用 table 变量建立基于存储过程的临时表分页算法描述如下:

设 T' 为一个以 RowNum 为主键并且该字段自增的一个临时表,有  $T' \subseteq T$ ; StartRow 为当前页记录中按照 RowNum 排序的最小值, StopRow 为最大值,规定第一页中的 StartRow = 0。

则:  $T' = \{x \mid x \in T, \text{Set RowCount @ StopRow, Order By ID}\}$

$E = \{x \mid x \in T', \text{RowNum} > = @ \text{StartRow, Order By RowNum}\}$

通过以上算法描述,其具体实现过程步骤如下:

1) 从输入参数给 @ StartRow 和 @ StopRow 进行相应的赋值;

2) 声明一个 table 变量 @ temp\_table, 其字段包括要查询的字段另加一个 RowNum 整型自增长的标识列;

3) Set RowCount @ StopRow, 控制向 @ temp\_table 插入的记录数;

4) 向 @ temp\_table 插入数据,其 SQL 语句描述为 "Insert @ temp\_table (要查询的字段名) Select 要查询的字段名 From 要查询的表 Where 查询条件";

5) 从 @ temp\_table 中获取 RowNum > = @ StartRow 的记录即是我们的最终结果,其 SQL 语句描述为 "Select \* From @ temp\_table Where RowNum > = @ StartRow Order By RowNum"。

### 2.2 基于存储过程的完全分页机制

这个方法的思想是,从全部符合条件的记录集中每次只获取一页的记录集发送到客户端。例如:符合条件的有 10 万条记录,而客户端每页可能仅显示 100 条记录。我们对总的结果集按主键 ID 进行升序排序,第一页则显示 ID 为 1 ~ 100 的记录,第二页显示 ID 为 101 ~ 200 的记录,如此下去,在用户点击“下一页”的时候,只在数据库中获得下 100 条记录即可。我们这里利用 Top 关键字,用于返回结果集中的前多少条记录数,然而没有更好的办法返回结果集中中间的一部分数据。

基于存储过程的完全分页算法描述如下:

设 操作 TOP (PageSize) 是获取结果集合中的前 PageSize 条记录;操作标志位为 Flag,其值为 1, 2, 3, 4 时分别代表首页、上页、下页、末页翻页操作;当前页面所要显示的记录数为 PageSize;当前页记录中排序字段最小值 FirstID, 当前页记录中排序字段最大值 LastID。

分页就是要从集合 T 中找到集合 E 的过程。基于存储过程的分页算法可表述为:

首页:  $E = \{x \mid x \in T, \text{flag} = 1, \text{TOP}(\text{PageSize}), \text{Order By ID}\}$

上页:  $E' = \{x \mid x \in T, \text{flag} = 2, \text{ID} < \text{FirstID}, \text{TOP}(\text{PageSize}), \text{Order By ID Desc}\}$

$E = \text{OrderByID}(E')$

下页:  $E = \{x \mid x \in T, \text{flag} = 3, \text{ID} > \text{LastID}, \text{TOP}(\text{PageSize}), \text{Order By ID}\}$

末页:  $E' = \{x \mid x \in T, \text{flag} = 4, \text{TOP}(\text{PageSize}), \text{Order By ID Desc}\}$

$E = \text{OrderByID}(E')$

由于在 SQL Server 2000 创建存储过程时, Top 关键字后面不能直接跟参数, 我们动态生成 SQL 语句字符串, 然后使用 exec 来执行改 SQL 字符串。通过以上算法描述, 其具体实现步骤如下:

1) 声明一个 varchar 变量 @strSQL, 用于存储动态生成的 SQL 语句。

2) 根据输入参数 @flag 来跳转执行相应的操作, 从而动态生成相应的 SQL 查询语句。有四类操作: “首页面”、“上一页”、“下一页”、“最后一页”。

3) 当 @flag = 1 (首页) 时: Set @strSQL = 'SELECT Top (@ pageSize) (要获取的字段名) FROM 表名 where 查询条件 order by 排序字段', 直接获取排序后结果集中的前 @ PageSize 条数据, 即是所需的首页的结果集。

4) 当 @flag = 2 (上一页) 时: Set @strSQL = 'Select \* From (Select Top (@ pageSize) 字段名 From 表名 where 查询条件 and 排序字段 < @ FirstID Order By 排序字段 Desc) As 别名 Order By 排序字段', 即首先按照排序字段进行降序排列, 获取排序字段值小于上一页中排序字段最小的前 @ PageSize 条记录。然后对这 @ PageSize 条记录再按照排序字段进行升序排列。此时返回的结果集正是需要的上一页的结果集。

5) 当 @flag = 3 (下一页) 时: Set @strSQL = 'Select Top (@ pageSize) 字段名 From 表名 where 查询条件 and 排序字段 > @ LastID Order By 排序字段', 即首先按照排序字段进行升序排列, 获取排序字段值大于当前页中排序字段最大的前 @ PageSize 条记录, 此时返回的结果集正是我们需要的下一页的结果集。

6) 当 @flag = 4 (末页) 时: Set @strSQL = 'Select \* From (Select Top (@ pageSize) 字段名 From 表名 where 查询条件 Order By 排序字段 Desc) As 别名 Order By 排序字段', 即首先按照排序字段进行降序排列, 获取前 @ PageSize 条记录。然后对这 @ PageSize 条记录再进行升序排列。此时返回的结果集正是我们需要的末页的结果集。

7) 执行动态生成的 SQL 语句即可。

### 3 两种分页机制的应用实例

我们为某大型企业开发了 Web 信息系统, 在类似报表查询系统中, 其查询数据很多来自于数据库中一个字段最多, 数据量极其庞大的工装派制表 (表名为 GZPZ\_Tab)。该表的主键为 GZPZ\_Id。以在制品清点账报表查询为实例来说明上述分页方法。例如要在工装派制表中查询某车间的在制品清点账数据。其查询条件是某车间 (字段名为 BM\_Name), 需获取产品类型 (XB\_Name)、工装号 (GZPZ\_GZH)、任务号 (GZPZ\_RWH)、派工数量 (GZPZ\_Count) 等字段 (为简单起见, 省去了更多的查询条件和字段)。

#### 3.1 基于 XML 分页的实现方法

结合基于 XML 的分页算法来实现对生产准备系统中在制品清点账报表的查询, 用 C# 语言描述, 大致过程如下:

```
DataSet myDataSet = new DataSet();
...
//数据库连接访问, 提取数据到 DataSet 的过程
XmlDataDocument datadoc = new XmlDataDocument(myDataSet);
//由 DataSet 创建 XmlDataDocument
datadoc.Save(Server.MapPath("文件路径 + XML 文件名"));
//生成 XML 文件
...
XmlDataDocument load(Server.MapPath
("文件路径 + XML 文件名")) //加载 XML 文件
XmlElement root ← XMLDataDocument 的根元素
```

```
XmlNodeList nodeList ← root.SelectNodes
("根据实际情况的 XPath 查询语句");
DataTable myTable ← MakeNewTable();
//根据要查询的字段名生成一个空的 DataTable
int flag = 0; //Flag 标志位, 初始化为 0
//用于标识所遍历到的节点是否超过本页要显示的结果集合
foreach (XmlNode node in nodeList)
{
    flag++;
    if (flag > (ViewState["当前页面的索引"] + 1) *
        每页显示的记录数)
        break;
    //如果遍历到的节点超出本页要显示的结果集时, 结束该遍历
    if (flag > ViewState["当前页面的索引"] * 每页显示的记录数)
    {
        DataRow myRow = myTable.NewRow();
        //建立一个定义好的数据行
        myRow["字段名"] = node.ChildNodes.Item
            (节点索引值).InnerText; //向字段填入数据
        ...
        //向 myRow 加载其余列的数据, 例如"GZPZ_GZH"、
        // "GZPZ_RWH"等
        myTable.Rows.Add(myRow); //将本行数据加载至 DataTable
    }
}
DataGrid1.DataSource = myTable;
//将 DataGrid 的数据源指定为 myTable
DataGrid1.DataBind(); //对 DataGrid 进行数据绑定
```

为了提高性能通过编写代码维护 ViewState 来实现分页。

设定 DataGrid 控件的 AllowCustomPaging 属性为 True, EnableViewState 属性为 False, 这样就不必在 ViewState 中存储没有必要的内容了, 释放了更多的内存资源。通过代码控制, 在 ViewState["CurrentPageIndex"] 中存储当前页的页码索引, 在点击下一页时, ViewState["CurrentPageIndex"] 的值递增 1 即可, 点击上一页时其值递减 1 即可。

#### 3.2 基于存储过程分页的现实方法

##### 3.2.1 基于存储过程的表变量分页

结合基于存储过程的临时表分页算法来实现对生产准备系统中在制品清点账报表的查询, 其关键部分如下:

```
输入参数: @ BM_Name as varchar(30) = null,
           @ StartRow as int = null, @ StopRow as int = null
主体部分: Declare @ temp_table table(
    [ RowNum] [int] IDENTITY (1, 1) Primary key NOT NULL,
    [ XB_Name] [varchar] (30),
    [ GZPZ_GZH] [varchar] (30),
    [ GZPZ_RWH] [varchar] (30),
    [ GZPZ_Count] [varchar] (30) )
Set RowCount @ StopRow
insert @ temp_table ([ XB_Name], [ GZPZ_GZH], [ GZPZ_RWH],
    [ GZPZ_Count]) SELECT XB_Name, GZPZ_GZH, GZPZ_RWH,
    GZPZ_Count FROM GZPZ_Tab where BM_Name = @ bm_name
order by GZPZ_Id
SELECT * FROM @ temp_table WHERE RowNum >= @ StartRow
Order by RowNum
```

该存储过程的第一个参数代表要查询的部门号。值得注意的是, 如果浏览者请求的页数太大, 页面性能也会下降。但是这样却有一个优点, 即可实现特定页的查询。例如, 如果需要直接查询第  $n$  页的数据 ( $n \geq 1$ ), 那么只需要在代码中通过 DataGrid 的 PageSize \*  $n$  计算出 @ StopRow, @ StartRow 则为 PageSize \* ( $n - 1$ ) + 1。

为追求性能和速度, 仍然设定 DataGrid 控件的 AllowCustomPaging 属性为 True, EnableViewState 属性为 False, 通过编写代码维护 ViewState。在“下一页”的事件中, 设置

ViewState 的过程描述为:

```
ViewState [ "StartRow" ] ←
  ViewState[ "StartRow" ] + DataGrid. PageSize;
ViewState [ "StopRow" ] ←
  ViewState [ "StartRow" ] + DataGrid. PageSize;
在“上一页”的事件中,设置 ViewState 的过程描述为:
ViewState [ "StartRow" ] ←
  ViewState[ "StartRow" ] - DataGrid. PageSize;
ViewState [ "StopRow" ] ←
  ViewState[ "StartRow" ] + DataGrid. PageSize;
```

### 3.2.2 基于存储过程的完全分页

结合基于存储过程的完全分页算法来实现对生产准备系统中在制品清点账报表的查询,该存储过程共有 5 个输入参数。存储过程关键部分如下:

```
输入参数定义: @bm_name varchar(30), @pageSize int,
@ FirstID nvarchar(20) = null, @ LastID nvarchar(20) = null,
@ flag int
存储过程主体部分:
Declare @ strSQL varchar(1000)
Set @ strSQL =
CASE @ flag
WHEN 1 THEN 'SELECT top' + CONVERT( varchar(30), @
  pageSize) + ' GZPZ_Id, XB_Name, GZPZ_GZH, GZPZ_
  RWH, GZPZ_Count FROM GZPZ_Tab where BM_Name = '
  + '''' + @bm_name + '''' + ' order by GZPZ_Id '
WHEN 2 THEN 'SELECT * from ( SELECT top ' + CONVERT
  ( varchar(30), @ pageSize) + ' GZPZ_Id, XB_Name, GZPZ_
  GZH, GZPZ_RWH, GZPZ_Count FROM GZPZ_Tab where
  BM_Name = ' + '''' + @bm_name + '''' + ' and GZPZ_Id < '
  + '''' + @ FirstID + '''' + ' order by GZPZ_Id desc) as bb
  order by GZPZ_Id'
WHEN 3 THEN 'SELECT top ' + CONVERT( varchar(30), @
  pageSize) + ' GZPZ_Id, XB_Name, GZPZ_GZH, GZPZ_
  RWH, GZPZ_Count FROM GZPZ_Tab where BM_Name = '
  + '''' + @bm_name + '''' + ' and GZPZ_Id > ' + '''' + @
  LastID + '''' + ' order by GZPZ_Id '
WHEN 4 THEN 'SELECT * from ( SELECT top ' + CONVERT
  ( varchar(30), @ pageSize) + ' GZPZ_Id, XB_Name, GZPZ_
  GZH, GZPZ_RWH, GZPZ_Count FROM GZPZ_Tab where
  BM_Name = ' + '''' + @bm_name + '''' + ' order by GZPZ_Id
  desc ) as aa order by GZPZ_Id'
END
exec( @ strSQL)
```

## 4 不同分页机制的测试比较

利用集成在 VS. NET 2003 中的 ACT( Application Center

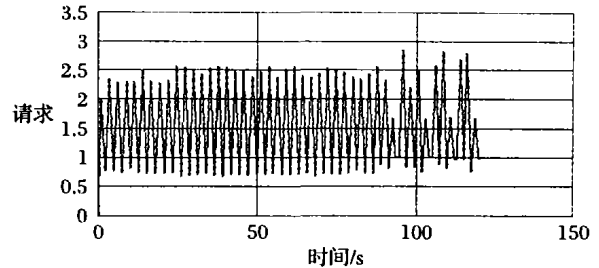
表 1 三种不同分页机制比较

分页机制	优点	缺点	适用范围
基于 XML 的分页机制	只访问一次数据库,减轻数据库压力,但需要创建 XmlDataDocument 等对象处理 XML	需要在 Web 服务器上先生成 XML 文件	适合于变动性不大的数据查询
基于存储过程的表变量分页机制	存储过程简单,可以实现特定页的查询	需多次和数据库交互,需在数据库建立变量,需插入当前页面以前的所有页面记录	适合海量结果集查询
基于存储过程的完全分页机制	只需获取当前一页的记录,只有 Select 查询操作	需多次和数据库交互,不能实现特定页查询	适合海量结果集查询

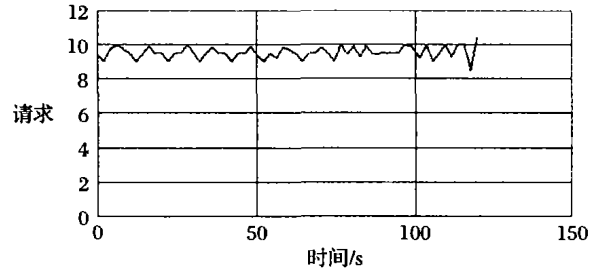
### 参考文献:

- [1] ARDESTANI K, HOFFMAN K, XIE D 高效掌握 ADO. NET——C# 编程篇[M]. 张哲峰,译. 北京:清华大学出版社,2003.
- [2] NGUYEN HQ. Web 应用测试[M]. 周志荣,姜南,译. 北京:电子工业出版社,2003.
- [3] ALLEN KS, AVERY J. ASP. NET 性能高级编程[M]. 侯斌,译. 北京:清华大学出版社,2003.
- [4] KNOWLES C, MOHR S. ASP. NET XML 高级编程[M]. 刘爽,译. 北京:清华大学出版社,2002.

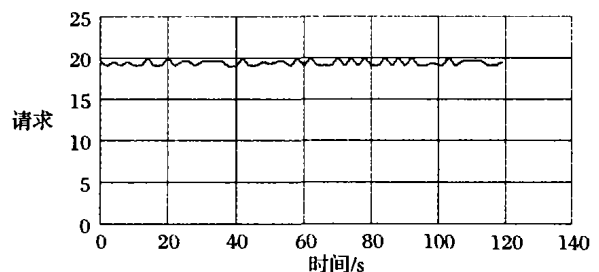
Test),在一台配置为 Dell Optiplex 160L 系列计算机上进行测试。在相同测试条件下,选取系统中一个最复杂的报表进行查询,测试每个页面的 RPS(每秒响应请求数)。



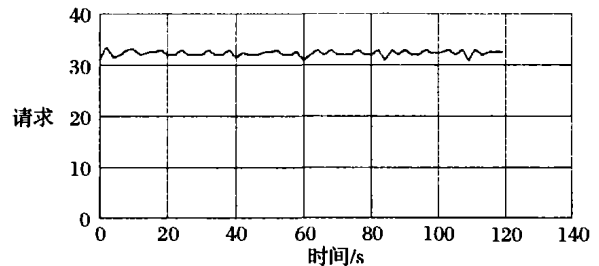
(a) 传统分页方法 (RPS 平均为 1.5)



(b) 基于 XML 分页方法 (RPS 平均为 9)



(c) 表变量分页机制 (RPS 平均为 19)



(d) 完全分页机制 (RPS 平均为 32)

图 3 测试结果

从测试数据可以看出,采用我们设计的分页机制可以大大提高系统的响应请求数,而且对于请求也显得更稳定,从而提高整个系统的性能和稳定性。表 1 是三种不同分页机制的比较。