Web Services for Peer-to-Peer Resource Discovery on the Grid

Domenico Talia and Paolo Trunfio

DEIS, University of Calabria
Via P. Bucci 41c, 87036 Rende, Italy
{talia,trunfio}@deis.unical.it

Abstract. Several features of today's Grids are based on centralized or hierarchical services. However, as Grid sizes increase, some of their functions should be decentralized to avoid bottlenecks and guarantee scalability. A way to provide Grid scalability is to adopt *Peer-to-Peer* (*P2P*) models and protocols to implement non hierarchical decentralized Grid services and systems. A core Grid functionality that could be effectively redesigned using the P2P model is *resource discovery*. This paper proposes an architecture for resource discovery that adopts a P2P approach to extend the model of the Globus Toolkit 3 information service. The *Open Grid Services Architecture* is exploited to define a *P2P Layer* of specialized Grid Services that support resource discovery across different Virtual Organizations in a P2P fashion. The paper discusses also a protocol, named *Gridnut*, designed for communication among Grid Services at the P2P Layer.

1 Introduction

The Grid computing model offers an effective way to build high-performance computing systems, allowing users to efficiently access and integrate geographically distributed computers, data, and applications. Several features of today's Grids are based on centralized or hierarchical services. However, as Grids used for complex applications increase their size from tens to thousands of nodes, it is necessary to decentralize their services to avoid bottlenecks and ensure scalability. As argued in [1] and [2], a way to provide Grid scalability is to adopt *Peer-to-Peer* (*P2P*) models and techniques to implement non-hierarchical decentralized Grid systems.

In the latest years, the Grid community has undertaken a development effort to align Grid technologies with Web Services. The *Open Grid Services Architecture* (*OGSA*) defines *Grid Services* as an extension of Web Services and lets developers integrate services and resources across distributed, heterogeneous, dynamic environments and communities [3]. Web Services define a technique for describing software components to be accessed, methods for accessing these components, and discovery methods that enable the identification of relevant service providers. Web Services and OGSA aim at interoperability between loosely coupled services independently from implementation, location or platform. Recently

the *Web Services Resource Framework* (*WSRF*) has been proposed for a more complete integration between Web and Grid Services [4]. OGSA defines standard mechanisms for creating, naming and discovering persistent and transient Grid Service instances, provides location transparency and multiple protocol bindings for service instances, and supports integration with underlying native platform facilities. The OGSA effort aims to define a common resource model that is an abstract representation of both real resources, such as processors, processes, disks, file systems, and logical resources. It provides some common operations and supports multiple underlying resource models representing resources as service instances. The OGSA model provides an opportunity to integrate P2P models in Grid environments since it offers an open cooperation model that allows Grid entities to be composed in a decentralized way.

A core Grid functionality that could be effectively redesigned using the P2P model is *resource discovery*. Resource discovery is a key issue in Grid environments, since applications are usually constructed by composing hardware and software resources that need to be discovered and selected. In the OGSA framework each resource is represented as a Grid Service, therefore resource discovery mainly deals with the problem of locating and querying information about useful Grid Services.

In *Globus Toolkit 3* (*GT3*) - the current implementation of the OGSA - information about resources is provided by *Index Services*. An Index Service is a Grid Service that holds information (called *Service Data*) about a set of Grid Services registered to it. A primary function of the Index Service is to provide an interface for querying aggregate views of Service Data collected from registered services. There is typically one Index Service per *Virtual Organization* (*VO*). When a VO consists of multiple large sites, very often each site runs its own Index Service that indexes the various resources available at that site. Then each of those Index Services is included in the VO's Index Service [5].

This paper proposes an architecture for resource discovery that adopts a P2P approach to extend the model of the GT3 information service. In particular, a *P2P Layer* of specialized Grid Services is defined to support discovery queries on Index Services of multiple VOs in a P2P fashion. The paper outlines also a modified Gnutella protocol, named *Gridnut*, designed for communication among Grid Services at the P2P Layer. Gridnut uses appropriate message buffering and merging techniques to make Grid Services effective as a way to exchange messages in a P2P fashion.

The remainder of the paper is organized as follows. Section 2 describes the architecture of the framework. Section 3 discusses the Gridnut approach and its performances. Finally, Section 4 concludes the paper.

2 The P2P Architecture

From the perspective of the GT3 information service, the Grid can be seen as a collection of VOs, each one indexed by a different Index Service. As mentioned before, Index Services of different sites can be included in a common higher-

level Index Service that holds information about all the underlying resources. However, for scalability reasons, a multi-level hierarchy of Index Services is not appropriate as a general infrastructure for resource discovery in large scale Grids. Whereas centralized or hierarchical approaches can be efficient to index resources structured in a given VO, they are inadequate to support discovery of resources that span across many independent VOs. The framework described here adopts the P2P model to support resource discovery across different VOs.

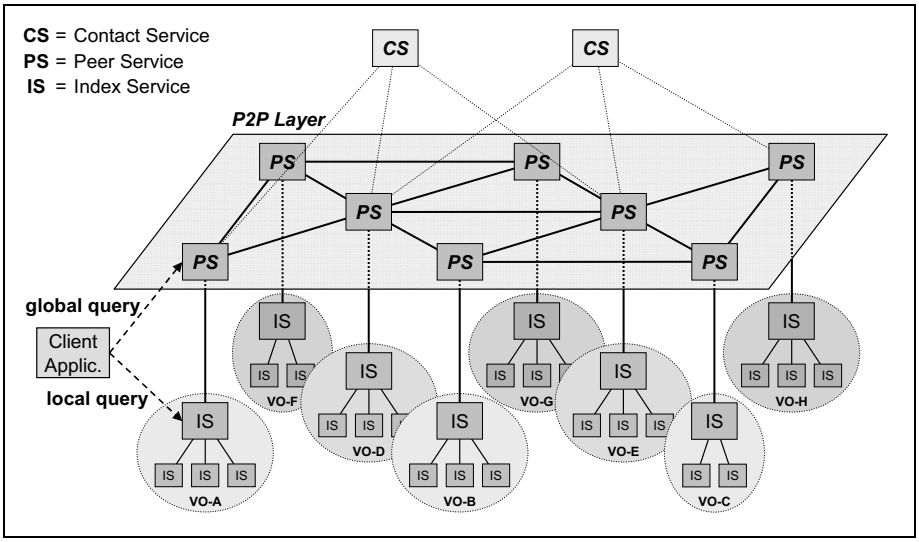

Fig. 1. Framework architecture.

Figure 1 shows the general architecture of the framework. Some independent VOs are represented; each VO provides one top-level *Index Service* (*IS*) and a number of lower-level Index Services.

A *P2P Layer* is defined on top of the Index Services' hierarchy. It includes two types of specialized Grid Services: *Peer Services* (*PS*), used to perform resource discovery, and *Contact Services* (*CS*), that support Peer Services to organize themselves in a P2P network.

There is one Peer Service per VO. Each Peer Service is *connected* with a set of Peer Services, and exchanges query/response messages with them in a P2P mode. The connected Peer Services are the *neighbors* of a Peer Service. A *connection* between two neighbors is a logical state that enables they to directly exchange messages. Direct communication is allowed only between neighbors. Therefore, a query message is sent by a Peer Service only to its neighbors, which in turn will forward that message to their neighbors. A query message is processed by a Peer Service by invoking the top-level Index Service of the corresponding VO.

A query response is sent back along the same path that carried the incoming query message.

To join the P2P network, a Peer Service must know the URL of at least one Peer Services to connect to. A convenient number of Contact Services is distributed in the Grid to support this issue. Contact Services cache the URLs of known Peer Services; a Peer Service may contact one or more well known Contact Services to obtain the URLs of registered Peer Services.

As shown in Figure 1, a *Client Application* can submit both *local* and *global* queries to the framework. A local query searches for information about resources in a given VO. It is performed by submitting the query to the Index Service of that VO. A global query aims to discover resources located in possibly different VOs, and is performed by submitting the query to a Peer Service at the P2P Layer. As mentioned before, the Peer Service processes that query internally (through the associated Index Service), and will forward it to its neighbors as in typical P2P networks.

The main difference between a hierarchical system and the framework described here is the management of global queries. Basically, in a hierarchical information service two alternative approaches can be used:

- the query is sent separately to all the top-level Index Services, that must be known by the user;
- the query is sent to one (possibly replicated) Index Service at the root of the hierarchy, that indexes all the Grid resources.

Both these approaches suffer scalability problems. In the P2P approach, conversely, global queries are managed by a layer of services that cooperate as peers. To submit a global query, a user need only to know the URL of a Peer Service in the Grid.

In the next subsection the design of the Peer Service and Contact Service components is discussed.

2.1 Services Design

Both Peer Service and Contact Service instances are identified by a globally unique *handle*. This handle is a URL - called *grid service handle* (*GSH*) in the OGSA terminology - that distinguishes a specific Grid Service instance from all other Grid Service instances.

The Peer Service supports four operations:

- ***connect***: invoked by a remote Peer Service to connect this Peer Service. The operation receives the *handle* of the requesting Peer Service and returns a *reject* response if the connection is not accepted (for instance, because the maximum number of connections has been reached).
- ***disconnect***: invoked by a remote Peer Service to disconnect this Peer Service. The operation receives the *handle* of the requesting Peer Service.
- ***deliver***: invoked by a connected Peer Service to deliver messages to this Peer Service. The operation receives the *handle* of the requesting Peer Service and an *array of messages* to be delivered to this Peer Service.

– ***query***: invoked by a client application to submit a global *query* to this Peer Service. Query responses are returned to the client through a notification mechanism.

The Contact Service supports one operation:

– ***getHandles***: invoked by a Peer Service to register itself and to get the handles of one or more registered Peer Services.

The *Web Services Description Language* (*WSDL*) is used in OGSA to expose the interfaces of Grid Services to remote clients. Figure 2, for instance, shows the Contact Service WSDL definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ContactService"
 targetNamespace="http://www.gridnut.org/namespaces/1.0/contact/ContactService"
 ... >
...
<gwsdl:portType name="ContactServicePortType" extends="ogsi:GridService">
   <operation name="getHandles">
      <input message="tns:GetHandlesInputMessage"/>
      <output message="tns:GetHandlesOutputMessage"/>
      <fault name="Fault" message="ogsi:FaultMessage"/>
   </operation>
</gwsdl:portType>
<message name="GetHandlesInputMessage">
   <part name="parameters" element="tns:getHandles"/>
</message>
<message name="GetHandlesOutputMessage">
   <part name="parameters" element="tns:getHandlesResponse"/>
</message>
<types>
<xsd:schema
 targetNamespace="http://www.gridnut.org/namespaces/1.0/contact/ContactService"
 ... >
  <xsd:element name="getHandles">
     <xsd:complexType>
        <xsd:sequence>
           <xsd:element name="handle" type="xsd:string"/>
           <xsd:element name="numHandles" type="xsd:int"/>
        </xsd:sequence>
     </xsd:complexType>
  </xsd:element>
  <xsd:element name="getHandlesResponse">
     <xsd:complexType>
        <xsd:sequence>
           <xsd:element name="getHandlesReturn" type="soapenc:Array"/>
        </xsd:sequence>
     </xsd:complexType>
  </xsd:element>
</xsd:schema>
</types>
</definitions>
```

Fig. 2. Contact Service WSDL definition.

Figure 3 and Figure 4 describe, respectively, the main software components of Peer Services and Contact Services.

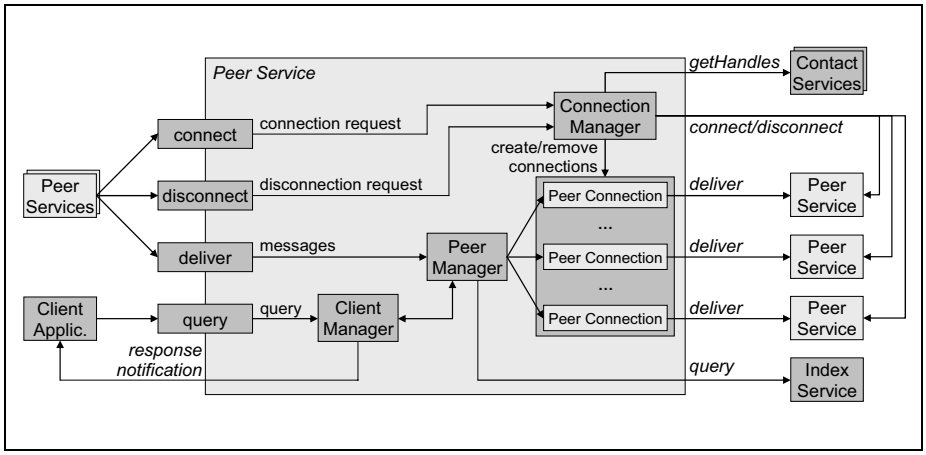

Fig. 3. Peer Service software components.

The Peer Service (see Figure 3) is composed by three main modules: *Connection Manager*, *Peer Manager*, and *Client Manager*.

The goal of the Connection Manager is to maintain a given number of connections with neighbor Peer Services. A *Peer Connection* object is used to manage the connection and the exchange of messages with a given Peer Service. A Peer Connection includes the *grid service reference* (*GSR*) of a given Peer Service, and a set of *transmission buffers* for the different kinds of messages directed to it. The Connection Manager both manages connection/disconnection requests from remote Peer Services, and performs connection/disconnection requests (as a client) to remote Peer Services. Moreover, it may invoke one or more Contact Services to obtain the handles of Peer Services to connect to.

The Peer Manager is the core component of the Peer Service. It both manages the messages delivered from other Peer Services, and interacts with the Client Manager component to manage client requests and to provide responses. It performs different operations on delivered messages: some messages are simply forwarded to one or more Peer Connections, whereas query messages need also a response (that in general is obtained by querying the local Index Service). Moreover, the Peer Manager generates and submits query messages to the network on the basis of the Client Manager requests.

The Client Manager manages the query requests submitted by client applications. It interacts with the Peer Manager component to submit the query to the network, and manages the delivery of query results to the client through a notification mechanism.

The Contact Service (see Figure 3) is composed by two software modules: *Contact Manager* and *Cache Manager*.

The Contact Manager manages the execution of the *getHandles* operation. Basically, it receives two parameters: the handle h of the invoker, and the number

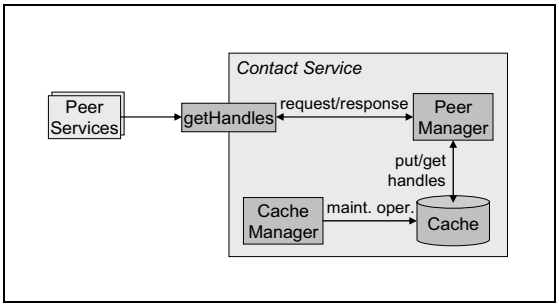

Fig. 4. Contact Service software components.

n of handles requested by the invoker. The Contact Manager first inserts (or updates) the handle h into a *Cache*, then it extracts (if available) n distinct handles from the Cache and returns them to the invoker. The handles can be extracted from the Cache on the basis of a given policy (e.g., randomly). If a Peer Service does not receive the requested number of handles, it can try to invoke the Contact Service later.

The Cache Manager performs maintenance operations on the Cache. For instance, it removes oldest (not recently updated) handles, performs content indexing, etc.

3 A P2P Grid Services-Based Protocol

Although Grid Services are appropriate for implementing loosely coupled P2P applications, they appear to be inefficient to support an intensive exchange of messages among tightly coupled peers. In fact, Grid Services operations are subject to an invocation overhead that can be significant both in terms of activation time and memory/processing consumption [6]. The number of Grid Service operations that a peer can efficiently manage in a given time interval depends strongly on that overhead. For this reason, standard P2P protocols based on a pervasive exchange of messages, such as Gnutella [7], are inappropriate on large OGSA Grids where a high number of communications take place among hosts.

To overcome this limitation, in [8] we proposed a modified Gnutella protocol, named *Gridnut*, which uses appropriate message buffering and merging techniques to make Grid Services effective as a way for exchanging messages among Grid nodes in a P2P fashion. The Gridnut protocol is designed for communication among Peer Services to support resource discovery across many independent VOs.

There are two main differences between Gnutella and Gridnut:

1. In Gnutella, messages are sent as a byte stream over TCP sockets, whereas in Gridnut messages are sent through a Grid Service invocation (by means of the Peer Service's *deliver* operation).

2. In Gnutella, each message is forwarded whenever it is received, whereas in Gridnut messages are buffered and merged to reduce the number of Grid Service invocations and routing operations executed by each Peer Service.

In particular, the basic principles adopted by Gridnut to reduce communication and routing overhead are

- *Message buffering*: messages to be delivered to the same Peer Service are buffered and sent in a single packet at regular time intervals.
- *Message merging*: messages with the same header (i.e, same type, identifier, and receiver) are merged into a single message with a cumulative body.

Similarly to Gnutella, Gridnut defines both a protocol to discover active Peer Services on the network, based on a *Ping/Pong* mechanism, and a protocol for searching the distributed network, based on a *Query/QueryHit* mechanism. We implemented and evaluated the Gridnut discovery protocol, even if we are also designing a general Gridnut search protocol.

In the following subsection we briefly compare the performance of Gridnut and Gnutella discovery protocols under different network and load conditions. Further details about performance measurements can be found in [8].

3.1 Performance Evaluation

The goal of our tests is to verify how significantly Gridnut reduces the workload - number of Grid Service operations - of each Peer Service. In doing this, we compared Gridnut and Gnutella by evaluating two parameters:

1. *ND*, the average number of *deliver* operations processed by a Peer Service to complete a discovery task. In particular, $ND = P \ / \ (N * T)$, where: P is the total number of *deliver* operations processed in the network, N is the number of Peer Services in the network, and T is the overall number of discovery tasks completed.
2. $ND(d)$, the average number of *deliver* operations processed by Peer Services that are at distance d from the Peer Service *S0* that started the discovery task. For instance: $ND(0)$ represents the number of *deliver* operations processed by *S0*; $ND(1)$ represents the number of *deliver* operations processed by a Peer Service distant one hop from *S0*.

Both *ND* and $ND(d)$ have been evaluated considering different network topologies. We distinguished the network topologies using a couple of numbers $\{N,C\}$, where N is the number of Peer Services in the network, and C is the number of Peer Services directly connected to each Peer Service (i.e., each Peer Service has exactly C neighbors).

Number of Deliver Operations

Figure 5 compares the values of *ND* in Gridnut and Gnutella in five network topologies: {10,2}, {30,3}, {50,4}, {70,4} and {90,4}, under different load conditions. A parameter R is used to define the load of the network. In particular,

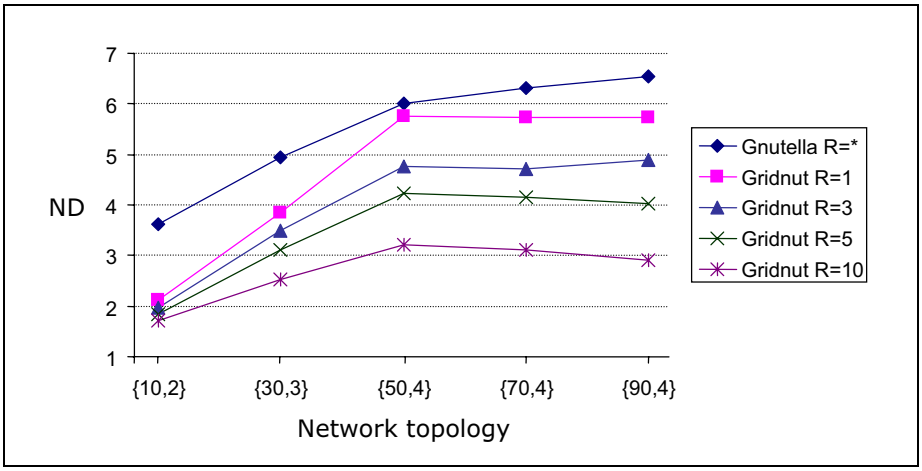

Fig. 5. ND versus the network topology.

R indicates the number of simultaneous discovery tasks that are initiated in the network at each given time interval. For Gridnut networks the values of *ND* when $R = 1$, 3, 5, and 10 are represented, whereas for Gnutella networks the average of the *ND* values measured when $R = 1$, 3, 5, and 10 is represented.

We can see that the number of *deliver* operations is lower with Gridnut in all the considered configurations. In particular, when the load of the network increases, the Gridnut strategy maintains the values of *ND* significantly low in comparison with Gnutella.

Distribution of Deliver Operations

Figure 6 compares the values of $ND(d)$ in Gridnut and Gnutella in five network topologies, with d ranging from *0* to *2*, and R fixed to *1*.

We can see that Gridnut implies a much better distribution of *deliver* operations among Peer Services in comparison with Gnutella. In Gnutella, the Peer Service that started the discovery task and its closest neighbors must process a number of Grid Service operations that becomes unsustainable when the size of the network increases to thousands of nodes. In Gridnut, conversely, the number of Grid Service operations processed by each Peer Service remains always in the order of the number of connections per peer.

This Gridnut behavior results in significantly lower discovery times since communication and computation overhead due to Grid Services invocations are considerably reduced as shown in Figure 6.

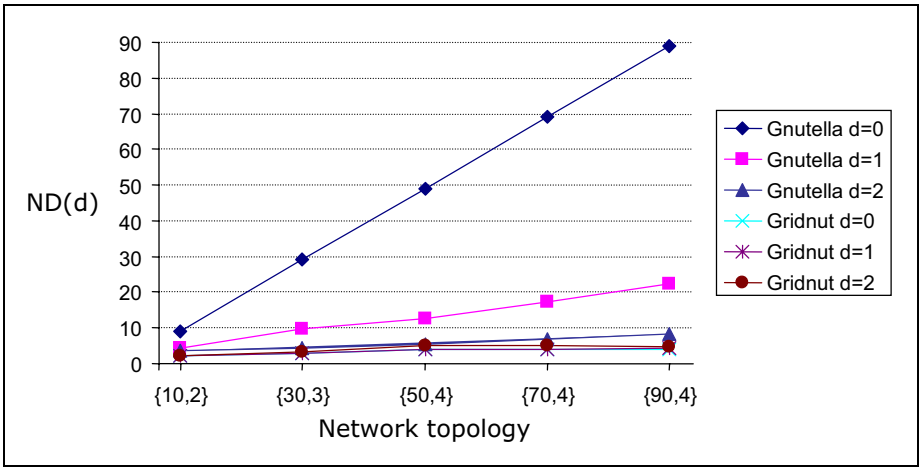

Fig. 6. ND(d) versus the network topology.

4 Conclusions

Resource discovery is a key issue in Grid environments, since applications are usually constructed by composing hardware and software resources that need to be discovered and selected. This paper proposed a framework for resource discovery that adopts a P2P approach to extend the model of the Globus Toolkit 3 information service. It defines a *P2P Layer* of specialized Grid Services that support resource discovery across different Virtual Organizations in a P2P fashion.

The paper discussed also a protocol, named *Gridnut*, designed for communication among Grid Services at the P2P Layer. Gridnut uses message buffering and merging techniques to make Grid Services effective as a way for exchanging messages among Grid nodes in a P2P mode. Experimental results show that appropriate message buffering and merging strategies produce significant performance improvements, both in terms of number and distribution of Grid Service operations processed.

Acknowledgements

This work has been partially funded by the Italian MIUR project "Grid.it" (RBNE01KNFP).

References

1. Foster, I., Iamnitchi, A.: On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. Proc. of the 2nd International Workshop on Peer-to-Peer Systems (2003)

2. Talia, D., Trunfio, P.: Toward a Synergy between P2P and Grids. IEEE Internet Computing, vol. 7, n. 4 (2003) 94-96
3. Foster, I., Kesselman, C., Nick, J. M., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. http://www.globus.org/research/papers/ogsa.pdf
4. Czajkowski, K. et al.: From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution. http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf
5. The Globus Alliance: MDS Functionality in GT3. http://www.globus.org/ogsa/releases/final/docs/infosvcs/4.html
6. The Globus Alliance: Globus Toolkit 3.0 - Performance Tuning Guide. http://www-unix.globus.org/toolkit/3.0/ogsa/docs/performance_guide.html
7. Clip2: The Gnutella Protocol Specification v.0.4. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf
8. Talia, D., Trunfio, P.: A P2P Grid Services-Based Protocol: Design and Evaluation. Proc. of the European Conference on Parallel Computing - EuroPar 2004 - (in press)