This is the author's version of a work that was submitted/accepted for publication in the following source:


This file was downloaded from: http://eprints.qut.edu.au/53748/

© Copyright 2012 IEEE

This work has been submitted to the IEEE for possible publication. Copyright may be transferred without notice, after which this version may no longer be accessible

Notice: Changes introduced as a result of publishing processes such as copy-editing and formatting may not be reflected in this document. For a definitive version of this work, please refer to the published source:
A Penalty-based Grouping Genetic Algorithm for Multiple Composite SaaS Components Clustering in Cloud

Zeratul Izzah Mohd Yusoh and Maolin Tang, Senior Member, IEEE
Science and Engineering Faculty
Queensland University of Technology
Brisbane, Australia
{z.mohdyusoh, m.tang}@qut.edu.au

Abstract—Software as a Service (SaaS) in Cloud is getting more and more significant among software users and providers recently. A SaaS that is delivered as composite application has many benefits including reduced delivery costs, flexible offers of the SaaS functions and decreased subscription cost for users. However, this approach has introduced a new problem in managing the resources allocated to the composite SaaS. The resource allocation that has been done at the initial stage may be overloaded or wasted due to the dynamic environment of a Cloud. A typical data center resource management usually triggers a placement reconfiguration for the SaaS in order to maintain its performance as well as to minimize the resource used. Existing approaches for this problem often ignore the underlying dependencies between SaaS components. In addition, the reconfiguration also has to comply with SaaS constraints in terms of its resource requirements, placement requirement as well as its SLA. To tackle the problem, this paper proposes a penalty-based Grouping Genetic Algorithm for multiple composite SaaS components clustering in Cloud. The main objective is to minimize the resource used by the SaaS by clustering its component without violating any constraint. Experimental results demonstrate the feasibility and the scalability of the proposed algorithm.

I. INTRODUCTION

Cloud computing [1] has become a popular computing paradigm in which scalable IT resources are provided services using Internet technology. One of the services offered is Software as a Service (SaaS) which refers to applications that are hosted by Cloud providers, as an alternative to the software packages that are usually installed locally in user’s machines [1]. SaaS is receiving growing interest from Cloud users in the recent years. Gartner Inc [2] forecasted that SaaS will continue to experience positive growth through 2015 with worldwide revenue projected to reach $22.1 billion. This illustrates that SaaS has been becoming more and more significant among software users and providers. Furthermore, advances in Cloud computing have made SaaS more accessible to a wide range of software users.

A SaaS can be delivered as composite application where it consists of a group of loosely-coupled individual applications that communicate with each other in order to form a higher-level functional system or application [3]. Delivering the SaaS in such an approach allows flexibility of the SaaS functionalities where components can be combined and recombined as needed. In addition, SaaS providers can reuse the components and which can reduce the SaaS delivery costs as well as decreased the subscription costs for its clients. However, this approach also introduces new challenges for SaaS resource management in a data centre.

A Cloud data centre usually consists of thousands of computation servers and storage servers with network links. Cloud data centre implement virtualization technology where a single physical server is sliced into a number of Virtual Machines (VMs) in which each of the VMs represent an execution environment in the Cloud. At the initial stage of the composite SaaS placement process, SaaS components are placed onto the physical servers. The components then are deployed in VMs for execution. The VM must have sufficient resources in order to fulfill the SaaS performance level specified in the user’s Service Level Agreement (SLA). In a dynamic environment of a Cloud data centre, where the workload of applications and resource capacities keep changing over time, the initial placement may need to be reconfigured in order to maintain the SaaS performance as well as to optimize the resources used. To achieve this, scheduled reconfiguration for the current component placement in VMs is triggered at certain period of time. However, the placement reconfiguration in existing resource management for Cloud data centre often ignores the communication or dependencies between SaaS components in their implementation. In a composite SaaS, these elements are important to be included as they will directly affect the performance of the SaaS. This paper will address this gap by proposing a solution for the reconfiguration placement of multiple composite SaaS in Cloud using clustering approach.

The remaining paper is organized as follows. Section 2 discusses related work. The problem formulation is described in Section 3. Section 4 presents the proposed algorithm. Then Section 5 is about the evaluation that has been carried out. The concluding remarks are presented in Section 6.

II. RELATED WORK

There is a large number of work addressing the problem of resource optimization in dynamic environment of Cloud data centres. The common objectives for optimization include minimizing the resource usage while maintaining the applications’
performances [4], [5], [6], minimizing the data center power consumption [7] and balancing the thermal distribution among the servers [8]. These objectives are achieved through various management plans at different levels. For instance, at the platform level, most existing works focus on the management of VM mapping to physical servers, while at the application level, the plan is to manage VM resources based on the application workload.

Existing works on resource management at the platform level use VM migration as the main approach in dealing with dynamic changes in the Cloud environment [5], [6]. The approach is used as it allows better utilization of resources at physical servers. Authors in [6] proposed a two-phase solution for VM migration in a data center, named Entropy. In the first phase, the minimum number of physical servers that can host the current VMs is determined. The second phase of the problem is concerned with finding the cheapest reconfiguration plan for the VM, based on the physical servers found in the first phase. The solution in this work is triggered by the current status of the VM. Other work like the one proposed in [5] triggers the migration periodically, based on its maintenance schedule. They proposed an algorithm that consists of four main processes: selection of the physical server that needs migration, selection of the suitable VM on that physical server, selection of the new physical server and assignment of the VM to the physical server. The selection is based on load profiles as well as behavior of the servers. All these works at a platform level consider a VM as an independent entity where it does not need to communicate to other VM or storage servers in completing its task. This paper proposes a different approach concerned with the communication involved between VMs and it will be tackled at the application level.

The communication among VMs is highlighted in [4] where the authors proposed a solution for reconfiguring placement that supports three types of constraints which are the VMs demands, communications and availability. The data center is modeled as a hierarchical structure that represents communication costs based on its hierarchy. Another work that is also concerned with communication among VMs is presented in [7]. In the paper, they considered a multi-tier application where the deployment may span multiple VMs. The proposed solution is designed at two levels. At the application level, there is a controller that will dynamically assign resources to applications based on their requirement, and at platform level, they propose a consolidation algorithm to re-map VMs to physical servers in the case of overload problem. The aim at the platform level is to optimize the data center power usage. Our work differs from all these solutions in the sense that they do not consider a composite application, in which a VM can host multiple components with different requirements. In addition, these components have to work with other components to achieve overall applications functionalities that subjects to user SLA. This paper will propose a solution to address this gap.

III. PROBLEM FORMULATION

At the initial stage of SaaS deployment, composite SaaS application and data components are placed on Cloud com-

utation servers and storage servers. Figure 1 illustrates an example of the initial placement, where the different shapes represent application components of a composite SaaS, and a VM can host multiple components at a time. A composite SaaS may span over multiple VMs and there are multiple composite SaaS hosted at a data center at a time.

Due to the dynamic environment of a data center, resources that have been initially allocated to SaaS application components may be overloaded or wasted. A typical data center usually schedules a placement reconfiguration of the components where this activity occurs at a certain period of time based on its need. Different approaches can be taken at different periods of time, and it can be done either dynamically or statically. Our approach is to deal with the dynamic environment at a static point of time, where a whole data center will be considered, in order to obtain a solution.

The placement reconfiguration for multiple composite SaaS is done by finding a new placement for the SaaS, by clustering suitable components together such that the new placement can minimize the resource used while satisfying the SaaS SLA. The problem inputs are: 1) the Cloud data centre, 2) the Cloud network tapology, and 3) the composite SaaS. The following sections describe the input and constraints in detail.

A. Cloud Data Center Modeling

A Cloud data center consists of computation servers and storage servers. Each server has its own resource capacities including processing capacity, memory size, secondary storage capacity and storage capacity. Each computation server has at least one virtual machine (VM), where the VM is given slices of the resources capacity of a computation server. A VM is associated with a cost, representing the capacity that the VM has. Table 1 summarizes the data center attributes.

B. Cloud Network Tapology

The Cloud network is represented by an undirected graph $G = (V, E)$. $V = \{CS \cup SS\}$ is the sets of vertices including physical servers and storage servers, $e \in E$ is the set of undirected edges connecting the vertices, if and only if there exists a physical link transmitting information from $v_i$ to $v_j$, where $v_i, v_j \in V$. $B_{v_i, v_j} : E \rightarrow \mathbb{R}^+$ and $L_{v_i, v_j} : E \rightarrow \mathbb{R}^+$ is a bandwidth and latency functions of the link from $v_i$ to $v_j$ respectively.

C. Composite SaaS Modeling

As mentioned earlier, there are multiple composite SaaS deployed in a Cloud data center at a time. Each of the
composite SaaS has its own application and data components with its minimum requirement for resources, as well as its SLA. In this paper, we will consider maximum response time with its minimum requirement for resources, as well as its composite SaaS has its own application and data components by the solution. The constraints are:

- A current placement configuration, \( P \), of application components \( AC \), onto virtual machines, \( VM \), given as:
  \[
P : AC \rightarrow VM, \text{ where } ac_{i,j} \mapsto P(ac_{i,j}) = vm_{x,y}
  \]

- A current location, \( L \), of the data components, \( DC \), at storage servers, \( SS \), given as:
  \[
  L : DC, \rightarrow SS \text{ where } dc_{i,q} \mapsto L(dc_{i,q}) = ss_k
  \]

### D. Problem’s Constraints

There are four types of constraints that need to be satisfied by the solution. The constraints are:

1. **Resource Constraints**: For all application components placed in a VM, the total requirements of the resources must not exceed VM’s resource capacity.

2. **Placement Constraints**: There are two types of placement constraints: a) anti-location constraint determining list of VM that should not be considered for hosting a specific component, and b) anti-colocation constraint determining list of application components that cannot be placed at a same VM. The solution must comply with the anti-location and anti-colocation constraint.

3. **Response time constraints**: The total execution time of a composite SaaS is calculated based on four numerical attributes which are: a) the time taken for transferring data between storage servers and virtual machine, b) the processing time of a component in a selected virtual machine, c) the execution time of a path in the SaaS workflow, d) the sum of the execution time of the critical path of each workflow multiplied by its weighing. All these attributes have been defined in our previous work \[12\]. Based on these four values, total execution time of the SaaS, \( TET \), is determined. The \( TET \) must not exceed the maximum response time of a SaaS as agreed in users’ SLA. This constraint is defined as below:

   \[
   TET(SC_i) \leq r_{SC_i}
   \]

4. **Sequence of migration constraints**: To change the placement from one VM to another, the solution has to consider the sequence of components that need to be moved based on the current placement at that time. Two scenarios will be considered in this problem:
   - **Sequential move**: A particular component can only be moved when another one has been completed. This is in the case of insufficient resources in the destination VM because it contains another component that due to be migrated.
   - **Cyclic move**: A set of components’ migration may need an intermediate destination machine. This is in the case of when two or more components need to be exchanged places.

Given all the input as above, the problem is to find a new placement of \( S \) onto \( VM \) by clustering the application components \( AC \), such that the placement will minimize the resources’ costs while satisfying the SaaS constraints. As component placement reconfiguration is an expensive process, the proposed solution will try to achieve the objective with a minimum number of changes to the current placement configuration.

### IV. THE PROPOSED ALGORITHM

As the approach for this problem is to cluster components into VMs, it suits naturally Grouping Genetic Algorithm (GGA) technique. GGA \[10\] is a modified version of Genetic Algorithm (GA) \[11\] where it is designed for solving grouping problems. While GA treats its chromosomes and cost function as a whole, GGA divides its chromosomes based on relevant groups and optimization of cost functions and genetic operations are done based on the grouping.

The problem has several constraints that a solution has to comply. A repair-based grouping GA (RGGA) for this problem

---

### Table I

**Sets and attributes of cloud resources**

<table>
<thead>
<tr>
<th>Cloud resources</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( cs_x \in CS )</td>
<td>The ( x^{th} ) computation server, ( cs_x ), in ( CS ), where ( CS ) is a set of ( k ) computation servers and ( 1 \leq x \leq k )</td>
</tr>
<tr>
<td>( ss_k \in SS )</td>
<td>The ( k^{th} ) storage server, ( ss_k ), in ( SS ), where ( SS ) is a set of ( r ) storage servers and ( 1 \leq k \leq r )</td>
</tr>
<tr>
<td>( vm_{x,y} \in VM )</td>
<td>The ( y^{th} ) virtual machine, ( vm ), for ( cs_x ) and ( VM ) is a set of all virtual machine, ( y \leq N )</td>
</tr>
</tbody>
</table>

### Table II

**SaaS modelling and requirements of composite SaaS**

<table>
<thead>
<tr>
<th>SaaS modelling</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( SC_i \subseteq S )</td>
<td>The ( i^{th} ) composite SaaS, ( SC_i ) in ( S ), ( S ) is a set of ( n ) composite SaaS, ( SC ), and ( 1 \leq i \leq n )</td>
</tr>
<tr>
<td>( ac_{i,j} \in AC )</td>
<td>The ( j^{th} ) application component, ( ac_{i,j} ) for ( SC_i ) and ( AC ) is a set of all application component, ( 1 \leq j \leq z )</td>
</tr>
<tr>
<td>( dc_{i,q} \in DC )</td>
<td>The ( q^{th} ) data component, ( dc_{i,q} ) for ( SC_i ) and ( DC ) is a set of all data component, ( 1 \leq q \leq x )</td>
</tr>
<tr>
<td>( wf_{i,p} \in WF )</td>
<td>A ( j^{th} ) business workflow for ( SC_i ) where ( WF \subseteq AC ), ( 1 \leq p \leq y )</td>
</tr>
<tr>
<td>( r_{SC_i} )</td>
<td>The maximum response time for ( SC_i )</td>
</tr>
<tr>
<td>( TS_{ac_{i,j}} )</td>
<td>Task size of ( ac_{i,j} )</td>
</tr>
<tr>
<td>( M_{ac_{i,j}} )</td>
<td>Memory requirement of ( ac_{i,j} )</td>
</tr>
<tr>
<td>( SZ_{ac_{i,j}} )</td>
<td>Size of ( ac_{i,j} )</td>
</tr>
<tr>
<td>( AD_{ac_{i,j}} )</td>
<td>Amount of read/write task of ( ac_{i,j} )</td>
</tr>
<tr>
<td>( W_{wf_{i,p}} )</td>
<td>Weighing for ( wf_{i,p} )</td>
</tr>
</tbody>
</table>
A. Chromosomes Representation

The chromosome is grouped based on composite SaaS in the Cloud. Each group has two compartments. The first compartment contains \( n \) genes, each of which corresponds to an application component in that particular group. The second compartment contains the ID of the virtual machine, where the application component would be placed in the new placement plan. Figure 2 shows an instance of the chromosome representation.

B. Genetic Operators

1) Crossover: The crossover operation is design based on the grouping chromosomes. A single point inter-group crossover will be used. This will combined segments from different SaaS, and produce two offsprings. The top two fittest among the parents and children are selected into the next generation.

2) Mutation: To promote further exploration in the search space, an inner-group mutation operator is used in order to keep the diversity of chromosomes in the population. The mutation operator is applied within a composite SaaS. It changes a virtual machine for a component to another virtual machine that also satisfy all the constraints.

C. Fitness Function

The aim of the problem is to create clusters of components of the multiple composite SaaS. Components that are clustered together will be placed onto a same server such that the new placement can minimize the resource’s costs while satisfying the SaaS constraints. The proposed solution will try to achieve this aim with a minimum number of migration from the current placement to the desired placement. These will be incorporated in the objective function of the problem. There are two parts of the objective function that will be used as a basis to evaluate each of the solutions.

\[ Y_x = ((F(TC) \times w_1) + (F(MC) \times w_2)) \]

where \( w_1 \) and \( w_2 \) are the weightage for each part, \( w_1 + w_2 = 0.5 \).

There are four constraints defined in Section 3.4. As each of the solution has to meet the components’ resource requirement before other constraints can be checked, solutions that violated resource constraint will be repaired. However, penalty will be imposed to solutions that violated placement constraint, \( V_1 \) or and time constraint \( V_2 \). For the sequence of migration constraint, it is incorporated in the changes cost of solutions in Section 4.3.2.

\[ TC = \sum_{vm_{x,y} \in VM} \text{Cost}_{vm_{x,y}} \]

where

\[ \text{Cost}_{vm_{x,y}} = \begin{cases} C_{vm_{x,y}} & \text{if } P(ac_{i,j}) = \text{vm}_{x,y} \text{ otherwise} \end{cases} \]

The following equation is to normalize \( TC \), and to ensure \( TC \) is less than the current placement cost:

\[ F(TC) = \begin{cases} 0, & \text{initialCost} - TC \geq \text{initialCost} \text{ otherwise} \end{cases} \]

2) The changes cost for a solution: Changing the current placement of a component from one VM to another requires some memory and bandwidth on both source and destination servers. These will incur some costs. To estimate this cost, the calculation for placement changes are based on the size of the component, \( Sz_{ac_{i,j}} \) as well as its memory requirement, \( M_{ac_{i,j}} \) as defined below:

\[ MC = \sum_{ac_{i,j} \in AC} \frac{Sz_{ac_{i,j}}}{\max(Sz_{AC})} \times 2 + \frac{M_{ac_{i,j}}}{\max(M_{AC})} \times 2 \]

The following equation to normalize MC:

\[ F(MC) = 1 - \frac{MC}{|AC|} \]

1) The cost of virtual machines used by the SaaS: VMs have their costs which is based on their resources’ capacity. To calculate the total cost of the virtual machines for a chromosome, the total VM cost to host the SaaS components, \( TC \), will be the basis of the evaluation. This is defined as:

The aim of the problem is to create clusters of components of the multiple composite SaaS. Components that are clustered together will be placed onto a same server such that the new placement can minimize the resource’s costs while satisfying the SaaS constraints. The proposed solution will try to achieve this aim with a minimum number of migration from the current placement to the desired placement. These will be incorporated in the objective function of the problem. There are two parts of the objective function that will be used as a basis to evaluate each of the solutions.

\[ Y_x = ((F(TC) \times w_1) + (F(MC) \times w_2)) \]

where \( w_1 \) and \( w_2 \) are the weightage for each part, \( w_1 + w_2 = 0.5 \).

There are four constraints defined in Section 3.4. As each of the solution has to meet the components’ resource requirement before other constraints can be checked, solutions that violated resource constraint will be repaired. However, penalty will be imposed to solutions that violated placement constraint, \( V_1 \) or and time constraint \( V_2 \). For the sequence of migration constraint, it is incorporated in the changes cost of solutions in Section 4.3.2.

\[ TC = \sum_{vm_{x,y} \in VM} \text{Cost}_{vm_{x,y}} \]

where

\[ \text{Cost}_{vm_{x,y}} = \begin{cases} C_{vm_{x,y}} & \text{if } P(ac_{i,j}) = \text{vm}_{x,y} \text{ otherwise} \end{cases} \]

The following equation is to normalize \( TC \), and to ensure \( TC \) is less than the current placement cost:

\[ F(TC) = \begin{cases} 0, & \text{initialCost} - TC \geq \text{initialCost} \text{ otherwise} \end{cases} \]

2) The changes cost for a solution: Changing the current placement of a component from one VM to another requires some memory and bandwidth on both source and destination servers. These will incur some costs. To estimate this cost, the calculation for placement changes are based on the size of the component, \( Sz_{ac_{i,j}} \) as well as its memory requirement, \( M_{ac_{i,j}} \) as defined below:

\[ MC = \sum_{ac_{i,j} \in AC} \frac{Sz_{ac_{i,j}}}{\max(Sz_{AC})} \times 2 + \frac{M_{ac_{i,j}}}{\max(M_{AC})} \times 2 \]

The following equation to normalize MC:

\[ F(MC) = 1 - \frac{MC}{|AC|} \]

The aim of the problem is to create clusters of components of the multiple composite SaaS. Components that are clustered together will be placed onto a same server such that the new placement can minimize the resource’s costs while satisfying the SaaS constraints. The proposed solution will try to achieve this aim with a minimum number of migration from the current placement to the desired placement. These will be incorporated in the objective function of the problem. There are two parts of the objective function that will be used as a basis to evaluate each of the solutions.
Algorithm 1: Penalty-based Grouping Genetic Algorithm

1  \textit{bestFitness} = 0
2  \textbf{randomly initialise} (Population)
3  \textbf{while} termination condition is not true \textbf{do}
4    \textbf{for} $X$ \in Population \textbf{do}
5      \textbf{if} $X$ violates SaaS resource requirements \textbf{constraint then}
6        \textit{Repair}(X)
7      \textbf{end if}
8      Calculate the new VM's cost
9      Calculate the cost of changing placement based on sequence of migration constraint
10     Calculate $X$ fitness value, $F(X)$, penalized if $X$ violates SaaS placement constraint and response time constraint
11     \textbf{if} $F(X) > \textit{bestFitness}$ \textbf{then}
12       \textit{Replace} bestFitness and store $X$
13     \textbf{end if}
14   \textbf{end for}
15  \textbf{Select individuals from the Population based on roulette wheel selection}
16  Probabilistically apply the crossover operator to generate new individual
17  Probabilistically select individuals for mutation
18  Use the new individuals to replace the old individuals in the Population
19 \textbf{end while}
20 \textbf{output} \textit{bestFitness}

V. EVALUATION

The experiment is conducted to evaluate the scalability and the quality of the solution of penalty-based GGA. Since there is no benchmark available for this problem, we compare the algorithm with a First Fit Decreasing (FFD) heuristic and the Repair-based Genetic Algorithm (RGGA) that have been developed before [13]. In the FFD, the VMs and SaaS components are sorted in decreasing order based on the capacity or requirement, and migrates each component to the first available VM. If the solution violates the time constraint, a new VM will be selected randomly and it will continue until the constraint is satisfied or until there is no improvement for 100 consecutive iterations. In the RGGA, each constraint violation will be repaired by generating a new value. All techniques are implemented in C++ and the experiments were carried out in desktop computers with 3 GHz Intel Core 2 Duo CPU and 4GB RAM.

For the experiment setting, we created five test cases of Cloud data centre that contained from 300 to 1500 VMs, with an increment of 300 for each test case. The attributes of the servers were randomly generated using the models presented in [14]. The composite SaaS is randomly created as well. We fixed the total SaaS in the Cloud at three, with the total of 15 application components and 6 data components. For PGGA and RGGA, the population size was 100, with crossover and mutation probabilities 0.95 and 0.05 respectively. The initial population is randomly generated and the termination condition is no improvement for the best individual in 25 consecutive generation. For the PGGA objective function, $w_1$ was set to 0.3 while $w_2$ is set to 0.2. These parameters were obtained through trials on randomly generated test problems. Parameters that led to the best performance in the trials were selected.

Considering the nature of all techniques, each of the test cases was repeated 10 times. Table IV shows the statistics of the experimental results including the best, worst, average and standard deviation of the VM costs for the PGGA, the RGGA and the FFD. Figure 3 shows the comparison in terms of VM costs. It can be seen that the PGGA can find better VM cost up to 24-34% less than the one proposed by FFD and 4-8% less than generated by RGGA in all test cases. This shows the benefit of the PGGA, in term of cost saving while maintaining the SaaS performance.

Figure 4 visualizes the average computation time for all techniques according to the test cases. Based on the result, PGGA has the longest computation time where it spent about 3-5 minutes to find a feasible solution while RGGA took about 1-2 minutes and FFD took less than one seconds for every
test cases. Although the time differences are significant, this is still affordable since the maintenance phase of the SaaS reconfiguration placement in the Cloud occurs at different time scales, from seconds to days, depending on the data centre’s needs.

VI. CONCLUSIONS

We have discussed a multiple composite SaaS component clustering problem for dynamic resource management in Cloud data centers. It involves reconfiguring the placement of the SaaS application components onto VMs that have varying sizes and costs. The major objective of the problem is to minimize the cost of the VM used by the SaaS while maintaining its performance. The problem aims to achieve the objective with minimum VM migration.

We proposed a penalty-based Grouping Genetic Algorithm for the problem. The algorithm considers the SaaS resource requirements, constraints as well as the communication requirements of the SaaS components, which are usually ignored in the traditional SaaS resource management. Based on the experimental results, the proposed algorithm always produces a feasible solution to all test problems and produced solutions that can reduce the overall cost of resources used. Although the computation time taken is quite long, it is still acceptable considering that there are various types of maintenance in a data center that are conducted at different time scales - in term of seconds to hours.

As for the future work, we plan to improve the algorithm computation time by decomposing the multiple composite SaaS into several subpopulations such that it can evolve independently in each generation while cutting down some processing time as this can be implemented in a parallel manner.

ACKNOWLEDGMENTS

This research was carried out as part of the activities of, and funded by the Smart Services Cooperative Research Centre (CRC) through the Australian Government’s CRC Programme (Department of Innovation, Industry, Science and Research).

REFERENCES