Zadatak D – Rešenje.

Listing 1 – Krajnja implementacija *JobServiceProvider* koja implementira traženi algoritam.

```
class JobServiceProvider : IJob
    {
        NetTcpBinding binding = new NetTcpBinding();
        private string internalEndpointName = "InternalRequest";

        public int DoCalculus(int to)
        {
            Trace.WriteLine(String.Format("DoCalculus method called - interval [1,{0}]",
to.ToString()), "Information");

            // all internal endpoints of all worker role processes not including this
worker role process
            List<EndpointAddress> internalEndpoints =
RoleEnvironment.Roles[RoleEnvironment.CurrentRoleInstance.Role.Name].Instances.Where(inst
ance => instance.Id != RoleEnvironment.CurrentRoleInstance.Id).Select(process => new
EndpointAddress(String.Format("net.tcp://{0}/{1}",
process.InstanceEndpoints[internalEndpointName].IPEndpoint.ToString(),
internalEndpointName))).ToList();

            int totalSum = 0;

            #region algorithm for dividing interval to equidistant subintervals
            int brotherInstances = internalEndpoints.Count;

            int result = to / brotherInstances;
            int remainder = to % brotherInstances;

            int lastParam = 0, currentBeginning = 0;

            Task<int>[] tasks = new Task<int>[brotherInstances];

            for (int i = 0; i < brotherInstances; i++)
            {
                currentBeginning = lastParam;
                lastParam += result;

                if (remainder > 0)
                {
                    lastParam++;
                    remainder--;
                }

                Trace.WriteLine(String.Format("Calling node at: {0}",
internalEndpoints[i].ToString()), "Information");

                int index = i;
                int openInterval = currentBeginning + 1;
                int closeInterval = lastParam;

                Task<int> calculatePartialSum = new Task<int>(() =>
                {
```

```
                    IPartialJob proxy = new ChannelFactory<IPartialJob>(binding,
internalEndpoints[index]).CreateChannel();
                    return proxy.DoSum(openInterval, closeInterval);
                });
                calculatePartialSum.Start();
                tasks[index] = calculatePartialSum;
            }
            #endregion

            Task.WaitAll(tasks);

            foreach (Task<int> task in tasks)
            {
                totalSum += task.Result;
            }

            return totalSum;
        }
    }
```